2007

NEW BIOINFORMATIC TECHNIQUES FOR THE ANALYSIS OF LARGE DATASETS

Justin Clay Harris
University of Kentucky, Clay.Harris@bucknell.edu

ABSTRACT OF DISSERTATION

Justin Clay Harris

The Graduate School

University of Kentucky

2007

NEW BIOINFORMATIC TECHNIQUES FOR THE ANALYSIS OF LARGE DATASETS

ABSTRACT OF DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
Department of Chemistry
at the University of Kentucky

By
Justin Clay Harris

Lexington, Kentucky

Director: Dr. Robert A. Lodder, Professor of Chemistry

Lexington, Kentucky

2007

ABSTRACT OF DISSERTATION

NEW BIOINFORMATIC TECHNIQUES FOR THE ANALYSIS OF LARGE DATASETS

A new era of chemical analysis is upon us. In the past, a small number of samples were selected from a population for use as a statistical representation of the entire population. More recently, advancements in data collection rate, computer memory, and processing speed have allowed entire populations to be sampled and analyzed. The result is massive amounts of data that *convey* relatively little information, even though they may *contain* a lot of information.

These large quantities of data have already begun to cause bottlenecks in areas such as genetics, drug development, and chemical imaging. The problem is straightforward: condense a large quantity of data into only the useful portions without ignoring or discarding anything important. Performing the condensation in the hardware of the instrument, before the data ever reach a computer is even better.

The research proposed tests the hypothesis that clusters of data may be rapidly identified by linear fitting of quantile-quantile plots produced from each principal component of principal component analysis. Integrated Sensing and Processing (ISP) is tested as a means of generating clusters of principal component scores from samples in a hyperspectral near-field scanning optical microscope. Distances from the centers of these multidimensional cluster centers to all other points in hyperspace can be calculated. The result is a novel digital staining technique for identifying anomalies in

hyperspectral microscopic and nanoscopic imaging of human atherosclerotic tissue. This general method can be applied to other analytical problems as well.

KEYWORDS: near-infrared, bioinformatics, biomedical spectroscopy, chemometrics, analytical

Multimedia Elements Used: JPEG (.jpg); Bitmap (.bmp); GIF (.gif).

Justin Clay Harris

July 13, 2007

NEW BIOINFORMATIC TECHNIQUES FOR THE ANALYSIS OF LARGE DATASETS

By

Justin Clay Harris

Robert A Lodder
Director of Dissertation

Robert B. Grossman
Director of Graduate Studies

July 13, 2007

RULES FOR THE USE OF DISSERTATIONS

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the dissertation in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this dissertation for use by its patrons is expected to secure the signature of each user.

Name	Date

DISSERTATION

Justin Clay Harris

The Graduate School

University of Kentucky

2007

NEW BIOINFORMATIC TECHNIQUES FOR THE ANALYSIS OF LARGE DATASETS

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
Department of Chemistry
at the University of Kentucky

By
Justin Clay Harris

Lexington, Kentucky

Director: Dr. Robert A. Lodder, Professor of Chemistry

Lexington, Kentucky

2007

For Tomas Henry Harris

In Memory of George Henry Arthur Harris

Acknowledgements

I would like to express my gratitude beyond words to all those who have generously offered their support, guidance, and patience. To both my immediate and extended family, you have been the support and encouragement that I have needed throughout my life. Special thanks go to my advisor, from whom I have learned the bounds of science are only limited by the human imagination.

TABLE OF CONTENTS

List of Tables

List of Figures

List of Abbreviations

AAA(s)	Abdominal Aortic Aneurysm(s)
ak	above the kidney branch
AngII	Angiotensin II
apoE	apolipoprotein E
bk	below the kidney branch
BNL	Brookhaven National Laboratory
c	control mouse
CLA	Cluster Analysis
COTS	Commercial Off-the-Shelf
CRISP	Complementary Randomized Integrated Sensing and Processing
DOE	Department of Energy
FFT	Fast Fourier Transform
HICI	Hyperspectral Integrated Computational Imaging
HT	Hadamard Transform
ICI	Integrated Computational Imaging
ICS	International Chemometrics Society
ISP	Integrated Sensing and Processing
LED(s)	Light Emitting Diode(s)
LSB	Lower Sideband Mode
MF(s)	Molecular Filter(s)
MFC	Molecular Factor Computing
MM	Moment Method
NIR	Near-Infrared
NSLS	National Synchrotron Light Source
NSOM	Near-field Scanning Optical Microscope
OCT	Optimal Cutting Temperature compound
PC(s)	Principal Component(s)
PCA	Principal Component Analysis
PDS	Piecewise Direct Standardization
QBEAST	Quantile Bootstrap Error-Adjusted Single-sample Technique
QQ	Quantile-Quantile
RTE	Radiative Transfer Equation
SAQQ	Spectral Analysis by Quantile-Quantile plotting
SCRISP	Shortened Complementary Randomized Integrated Sensing and Processing
SD(s)	Standard Deviation(s)
SSSI	Solid State Spectral Imager
SSSI-I	First Solid State Spectral Imager
SSSI-II	Second Solid State Spectral Imager
VUV	Vacuum Ultraviolet
x	experimental mouse

List of Files

Figure #	File name	Size
Figure 3.1	Figure2.1.jpg	2,534,340 bytes
Figure 3.2	Figure2.2.bmp	3,497,526 bytes
Figure 3.3	Figure2.3.gif	10,348 bytes
Figure 3.4	Figure2.4.gif	2,472 bytes
Figure 3.5	Figure2.5.jpg	255,433 bytes
Figure 3.6	Figure2.6.jpg	45,882 bytes
Figure 3.7	Figure2.7.bmp	271,430 bytes
Figure 3.8	Figure2.8.gif	5,228 bytes
Figure 3.9	Figure2.9.gif	40,621 bytes
Figure 3.10	Figure2.10.gif	47,072 bytes
Figure 3.11	Figure2.11.jpg	1,245,882 bytes
Figure 3.12	Figure2.12.gif	44,151 bytes
Figure 3.13	Figure2.13.gif	5,846 bytes
Figure 3.14	Figure2.14.gif	53,335 bytes
Figure 4.1	Figure3.1.jpg	59,140 bytes
Figure 4.2	Figure3.2.gif	14,271 bytes
Figure 4.3	Figure3.3.jpg	336,761 bytes
Figure 4.4	Figure3.4.jpg	1,443,934 bytes
Figure 4.5	Figure3.5.gif	8,342 bytes
Figure 4.6	Figure3.6.jpg	472,401 bytes
Figure 4.7	Figure3.7.jpg	1,500,650 bytes
Figure 4.8	Figure3.8.gif	1,757 bytes
Figure 4.9	Figure3.9.gif	7,471 bytes
Figure 4.10	Figure3.10.jpg	419,204 bytes
Figure 4.11	Figure3.11.jpg	101,476 bytes
Figure 4.12	Figure3.12.jpg	203,139 bytes
Figure 4.13	Figure3.13.jpg	171,308 bytes
Figure 4.14	Figure3.14.jpg	615,644 bytes
Figure 4.15	Figure3.15.jpg	800,431 bytes
Figure 4.16	Figure3.16.jpg	583,540 bytes
Figure 4.17	Figure3.17.jpg	1,009,621 bytes
Figure 4.18	Figure3.18.gif	4,372 bytes
Figure 4.19	Figure3.19.gif	3,248 bytes
Figure 4.20	Figure3.20.jpg	1,615,202 bytes
Figure 4.21	Figure3.21.jpg	3,216,025 bytes
Figure 4.22	Figure3.22.gif	55,027 bytes
Figure 4.23	Figure3.23.jpg	572,631 bytes
Figure 4.24	Figure3.24.jpg	175,074 bytes
Figure 4.25	Figure3.25.gif	80,536 bytes
Figure 4.26	Figure3.26.gif	9,445 bytes
Figure 4.27	Figure3.27.jpg	46,410 bytes
Figure 4.28	Figure3.28.gif	60,126 bytes
Figure 4.29	Figure3.29.gif	16,356 bytes

Figure 4.30	Figure3.30.gif	6,053 bytes
Figure 4.31	Figure3.31.gif	16,471 bytes
Figure 4.32	Figure3.32.bmp	1,158,198 bytes
Figure 4.33	Figure3.33.bmp	1,158,198 bytes
Figure 4.34	Figure3.34.bmp	481,078 bytes
Figure 4.35	Figure3.35.bmp	481,078 bytes
Figure 5.1	Figure4.1.gif	21,119 bytes
Figure 5.2	Figure4.2.gif	32,494 bytes
Figure 5.3	Figure4.3.gif	15,486 bytes
Figure 5.4	Figure4.4.gif	25,264 bytes
Figure 5.5	Figure4.5.gif	12,674 bytes
Figure 5.6	Figure4.6.gif	17,337 bytes
Figure 5.7	Figure4.7.gif	13,668 bytes
Figure 5.8	Figure4.8.gif	22,260 bytes
Figure 5.9	Figure4.9.gif	13,255 bytes
Figure 5.10	Figure4.10.jpg	835,451 bytes
Figure 5.11	Figure4.11.jpg	781,311 bytes
Figure 5.12	Figure4.12.jpg	858,805 bytes
Figure 5.13	Figure4.13.jpg	826,005 bytes
Figure 5.14	Figure4.14.bmp	850,134 bytes
Figure 5.15	Figure4.15.bmp	2,510,334 bytes
Figure 5.16	Figure4.16.bmp	4,206,582 bytes
Figure 5.17	Figure4.17.jpg	258,284 bytes
Figure 5.18	Figure4.18.bmp	88,212 bytes
Figure 5.19	Figure4.19.jpg	121,405 bytes
Figure 5.20	Figure4.20.bmp	481,078 bytes
Figure 5.21	Figure4.21.bmp	481,078 bytes
Figure B.1	FigureB.1.jpg	583,540 bytes
Figure B.2	FigureB.2.gif	37,173 bytes
Figure B.3	FigureB.3.gif	34,006 bytes
Figure B.4	FigureB.4.gif	5,936 bytes
Figure B.5	FigureB.5.gif	146,069 bytes
Figure B.6	FigureB.6.jpg	69,072 bytes
Figure C.1	FigureC.1.jpg	855,189 bytes
Figure C.2	FigureC.2.jpg	835,451 bytes
Figure C.3	FigureC.3.jpg	781,311 bytes
Figure C.4	FigureC.4.jpg	858,805 bytes
Figure C.5	FigureC.5.jpg	826,005 bytes
Figure C.6	FigureC.6.jpg	843,038 bytes
Figure C.7	FigureC.7.gif	108,955 bytes
Figure C.8	FigureC.8.gif	50,025 bytes
Figure C.9	FigureC.9.gif	42,585 bytes
Figure C.10	FigureC.10.gif	62,208 bytes
Figure C.11	FigureC.11.gif	1,416 bytes
Figure C.12	FigureC.12.gif	54,758 bytes
Figure C.13	FigureC.13.gif	21,326 bytes

Figure C.14	FigureC.14.gif	12,433 bytes
Figure C.15	FigureC.15.gif	29,379 bytes
Figure C.16	FigureC.16.gif	37,173 bytes
Figure C.17	FigureC.17.gif	34,006 bytes
Figure C.18	FigureC.18.gif	5,936 bytes
Figure C.19	FigureC.19.gif	87,357 bytes
Figure C.20	FigureC.20.gif	46,825 bytes

Chapter One - Introduction

Motivation

Additions to the wealth of human knowledge fuels research and innovation in general. Novel analytical data analysis techniques as well as novel analytical instrumentation are of tremendous interest to scientists in many fields of research and innovation. In particular, the ability to analyze samples *in situ* or with minimal sample preparation is of utmost importance to not only decrease expense but also to decrease the time required for analysis. This becomes even more important when dealing with cases of disease in humans undergoing various types of treatment. In addition, there is a lack of rapid data analysis techniques for reproduction of digital images whose color is representative of chemical composition. Specifically, the field of bioinformatics lacks methods for rapidly staining digital images based upon chemical composition. Digital staining is a process where digital images are color coded to graphically represent features of interest, such as chemical species concentration. This dissertation presents a novel method for the analytical analysis of hyperspectral images as well as its application to a novel solid state spectrometer with no moving parts.

In some cases, the data collection rate of hyperspectral imaging vastly exceeds the data processing rate. In addition, high-speed applications require additional throughput for any usefulness of the individual spectroscopic technique. In these situations, the analysis process must be taken to the hardware of the instrument itself. In otherwords, the calculations on data collected from the detector that originally took place in a computer

are now done in the hardware of the instrument before the signal is collected by the detector. Integrated sensing and processing (ISP) is a solution to the problems of data processing time and spectroscopic improvements for individual applications. In this dissertation, an ISP approach called molecular factor computing (MFC) is applied to the near-field scanning optical microscope (NSOM) throughput problem. Extensive time was spent at the National Synchrotron Light Source (NSLS) at Brookhaven National Laboratory (BNL) coupling the NSOM to the intense synchrotron light source. The combination of MFC-NSOM with the synchrotron has led to a substantial reduction in scanning times. Chemical information at subwavelength spatial resolutions within living cells could be obtained in real time.

The goal of this work was twofold: (1) to develop a new technique for rapidly identifying chemical populations from spectroscopic data of numerous types and (2) to develop a new technique for a two-dimensional image reconstruction based upon the chemical populations identified. These images enable similarities and differences within the sample to be visually apparent in the digitally stained images produced. Ultimately, this technique is applied through MFC to create an NSOM that creates digitally stained images based upon the Molecular Filters (MFs) used.

Content Summary

The research presented in this dissertation was arranged to allow the reader to transition smoothly from one chapter to the next. However, each chapter is a publication under development and is intended to give the avid reader the information necessary to

understand the research preformed. Chapter Two was included to give most readers the background knowledge necessary to fully engage themselves with the research presented. Chapter Two is composed of three main sections: Hyperspectral Imaging Calibration, Chemometrics, and Molecular Factor Computing (MFC). The Hyperspectral Imaging Calibration section is being developed for a chapter in the book Comprehensive Chemometrics. This section gives an overview of hyperspectral imaging followed by details on internal and external spectral calibration. The Chemometrics section gives brief details on Principle Component Analysis (PCA). Chapter Two concludes with a section introducing the concepts of Molecular Factor Computing.

Chapters Three through Five of the dissertation represent the novel research conducted. They include: Chapter Three - Imaging Bioinformatics Using Cluster Analysis with Quantile-Quantile Plots and Synchrotron IR Microspectrometry for the Study of Abdominal Aortic Aneurysm in the ApoE -/- Mouse; Chapter Four - Application of Spectral Analysis by Quantile-Quantile Plots to Data Collected from a Novel Solid State Spectral Imager (SSSI); and Chapter Five - Integrated Computational Imaging with a Near-Infrared Near-field Scanning Optical Microscope (ICI NIR-NSOM). Chapter Three introduces a new chemometric technique, shows that it performs well, and gives an example of its functionality. This approach is being used to improve our understanding of Abdominal Aortic Aneurysms and their chemical precursors. Chapter Four introduces a novel spectrometer with no moving parts capable of producing high spatial and spectral resolution images at ten meters. The prototype SSSI was designed specifically for a mission to Mars. It could also be applied to various other “extreme environment”

problems. Finally, Chapter Five presents a new NIR-NSOM capable of doing integrated computational imaging within the instrument. This instrument has the ability to spectroscopically determine cellular kinetics at sub wavelength resolution. In the conclusion to this dissertation, various projects that could evolve from the research herein are discussed. Following the conclusion is a comprehensive list of MatLab functions and manuals for the instruments that were developed in the research presented in this dissertation.

Chapter Two – Hyperspectral Imaging Calibration, Chemometrics, and Molecular Factor Computing

Hyperspectral Imaging Calibration

Introduction.

The field of spectroscopy has grown and expanded over the years, and the variety of applications it now spans include weather maps, geography, medical research, satellite imagery of the earth and other planets, and even meat and produce control. The current model is that light (electromagnetic radiation) travels from a source until it interacts with a substance. The substance will ultimately absorb, transmit, scatter, and/or reflect a portion of the light. The reflected light contains different wavelengths and intensities (energies). A spectrometer is an instrument that quantifies electromagnetic radiation in terms of intensity and frequency. The identifiable quality of a spectrometer is that it evaluates light as a function of wavelength or frequency. A prism or grating is used to separate the wavelengths before the energy or intensity is measured[1, 2].

All materials absorb and reflect light. These phenomena create the great variety and intensity of color in our everyday lives. For different materials the frequency and wavelength of light absorbed and emitted is diverse, and thus creates a unique spectrum for every known substance. In the lab, the angle of incidence, angle of reflectance, angle of refraction, and incident energy can all be controlled. The source of the incident radiation, whether it is a laser, Xenon lamp, Nernst glower, etc., is specifically chosen for the sample being measured and the spectrum sought. Incident energy is controlled by

virtue of choosing the source. The angle of incidence describes the angle at which incident radiation strikes the sample and is easily controlled. Therefore, the angle of specular reflectance can be known and measured. In many spectroscopy applications the energy source and reflectance cannot be controlled with any certainty. Remember that each material absorbs, transmits, and reflects a distinctive spectrum. There are some materials that generate a much more distinctive spectrum in certain regions of the spectrum. The spectral signature of some materials in the ultraviolet (UV) region could be very similar, but the same materials could have very different spectra in the X-ray region[2, 3].

Take a moment to think about the concept of an image. An image is considered to be a representation of a subject. Therefore, a spectral image is the representation of the spectrum of subject. Fundamental to the field of spectroscopy is the perception of a pixel (abbreviation from the first letters of picture element). A pixel is the single most basic element of an image; pixels are combined in large quantities to form an image. A pixel usually has a 1, 3, or 4 component display setting: grayscale, red-green-blue (RGB), or cyan, yellow, magenta, and black (CYMK). Each pixel is also defined by the number of bits it contains, or bits per pixel (bpp). Bits represent binary code; they are either a 1 or a 0. For instance, an 8-bit pixel has 2^8 shades of color. It is not unusual for pixels to have millions (2^{20}) of shades of colors. So, a RBG pixel at 8-bits has 256 intensities to represent each of three colors in a pixel[4]. In a hyperspectral image, a pixel is a representation of the intensity values from a single detector. Each wavelength collected will have an intensity value. Generally, assuming the sensitivity of the detector stays the

same, the more pixels that are represented in an image of a sample (higher spatial resolution), the closer the image will appear to the sample[2, 5]. Therefore, a spectral image is the representation of an object by a spectrum for each pixel within the image. Spectral images are obtained by combining two technologies, spectroscopy and conventional/remote imaging[2, 4, 5].

Remote imaging is the taking of an image of an object from a remote position, without interacting physically with it. There are two types, passive and active. Active remote imaging involves a source transmitting electromagnetic radiation that interacts with the sample before returning, as in radar. Passive remote imaging involves using natural light, such as the sun's rays or moonlight as a source. Examples for passive imaging include the video camera and a satellite; radar and a scanning electron microscope would be examples of active imaging. In this chapter, the term spectral block will be used as a predetermined set of wavelengths, and spectral region will be a broader set of wavelengths such as the infrared, visible, or ultraviolet spectral regions. The difficulty in combining spectroscopy and remote imaging is that spectroscopy generally requires a detector and source for each spectral region. A ground-based spectrometer can have detectors and sources that are manipulated, changed, replaced, and/or may be large in size. In space, it is much more complex as satellites are not easily manipulated after being launched[1].

Spectral imaging utilizes a combination of spectroscopy and conventional/remote imaging to provide a complete spectrum for a single pixel. There are three types of

spectral imaging: multispectral, hyperspectral, and ultraspectral. Multispectral imaging provides images using only narrow wavelengths of light. In other words, it detects one spectral block, but that block is obtained by separating it into a few large portions. A discontinuous spectrum is created. The detection process is not able to differentiate between specific wavelengths, as adjacent wavelengths are lumped together. The portions of wavelengths are represented in an image by mixing a set of different colors and shades of colors. Each wavelength is represented by a unique color and/or shade. The primary point to note is that multispectral imaging does not produce a complete spectrum of a spectral region[3, 6-9].

Hyperspectral imaging is different compared to multispectral in that the images that it produces are not as disjointed, but are more contiguous. The spectrum for every pixel is scanned using hundreds of adjacent, very narrow bands. A great deal more information can be gathered compared to multispectral imaging because of the additional bands. The advantage to this is that the spectrum can be collected, saved, and analyzed at a later date. Sometimes a representational image is created by taking the average of all the intensity values for one pixel, and a total absorbance value for that pixel is created. When placed next to the total absorbance values of all the other pixels, a representative image is created and can be analyzed later. You may be wondering why an image needs to be analyzed once it is created. Although the image is created from the average intensity value from the spectrum, correction for the change in incident angle and incident energy from the source and detector need to be completed. Atmospheric interference and noise will also need to be eliminated. Another problem that arises is that the entire spectrum

taken for millions of pixels generates quite a significant amount of recorded data. While it is the quantity of spectral information gathered that allows the additional discrimination of distinctive materials, the processing and memory required can be excessive. Data compression assists in processing time and will be discussed later[1, 5, 8, 10, 11].

Ultraspectral imaging is considered to be an advancement of hyperspectral imaging because it allows for even finer spectral resolution due to the narrower bands collected and the added channels[6]. There is the potential for thousands of bands to be collected at once. Ultraspectral imaging has also been defined as the collection of hyperspectral images adjacent to each other, allowing for the boundaries between the spectral blocks to no longer be a substantial problem. There is still much more progress that needs done in this field[6]. Although it is not frequently quantized in this manner, Bianco *et al* differentiates the three types in terms of bands and spectral resolution[6]. They state that multispectral, hyperspectral, and ultraspectral have 10-50, 50-1000, and 10-100 bands respectively, with a spectral resolution of 0.1, 0.01, and 0.001 respectively[6].

The next concept that needs to be understood underlines how a computer program can differentiate between thousands or even millions of spectra. A hyperspectral image can contain a spectrum of hundreds or thousands of adjacent spectral bands for each pixel. Each of these bands has a measured amount or element of reflectance. Consider, for instance, that each reflectance value is given a numerical value. The reflectance values may be plotted versus the wavelength of light reflected, creating a reflectance spectrum. In infinite dimensional spectral space, a unique reflectance spectrum is generated for

every material. A computer can deal with multidimensional mathematics quite easily, and thus can compare and contrast millions of unique spectra to each other. The comparison of spectra is then able to reach new specificity because they can be compared in many different dimensions. A single hyperspectral image is essentially a 3-D image with the x and y axis as spatial coordinates and the z axis as the spectral channels. So, a single pixel in a hyperspectral image contains a 2-D spatial image with hundreds of 1-D spectral images "stacked" on top. This creates what could be called a 3-D hyperspectral data cube, or hypercube[1, 5, 7, 8, 10, 13-15].

Calibration is one of the challenges that face hyperspectral image analysis. Calibration of an instrument involves establishing the association of the output (reflectance) measured by the instrument to the input (electromagnetic radiation). Ordinarily, in ground-based instruments, this is a comparably a simple process. But for remote sensing, the challenges are more complex. First, the distance between the sensor and the material in question may vary significantly in remote sensing. For example, a satellite orbiting a planet could measure the output electromagnetic radiation from the sun and compare it to the electromagnetic radiation reflected from an analyte on the planet's surface. The light could interact with molecules in the atmosphere, clouds, perhaps even water on the surface of the planet, before it interacts with the analyte of interest. The material's unique reflectance spectra could interact with just as many but perhaps different compounds before it is detected by the sensor onboard the satellite. Even the angle of incidence of the sun's rays and the reflectance angle changes with respect to time. So, obtaining an appropriate and accurate background for hyperspectral images can be

challenging. Complicated algorithms and predictions are needed in order to calibrate spectral imagers correctly[6-8, 16].

Linear regression modeling is a common mathematical technique used to describe a relationship between two or three variables of interest. While not implying causality, a linear regression line can demonstrate association between the different variables. As defined earlier, remote sensing is the ability to measure data without actual contact with the sample[7]. The difficulty lies in that, like any scientific data collected, the data needs to have both accuracy and precision. Accuracy can be explained as the ability to compare a data point or set of data points to one another because they both adhere to a recognized standard. Precision is the ability to duplicate the data points found again and again. There must be a background or calibration curve that allows the data being collected to be compared to other standards. In other words, the conditions under which data is collected must be the same or shown to be not affecting the data being collected. As an example, wear and tear on a plane and a car cannot be compared to each other, because the circumstances are completely different. The airplane moves at a different speed, a different altitude, different pressure, etc. The background or collection conditions are vastly different and have a large affect on the data being collected. The challenge is that hyperspectral imaging generates immense amounts of data about multiple locations, at multiple times. If multiple sensors are being used to collect data, they all must have similar backgrounds or have been calibrated so that each sensor provides a comparable result. According to G. Asrar, all instruments must adhere to standards set by the

National Bureau of Standards (NBS) in order for the results to be accepted by the scientific community[7, 8].

There are three parameters that are applicable to calibration. They are calibration of the intensity or amplitude of signal, calibration of the spatial aspect, and finally, the spectral calibration. The intensity calibration basically insures that the amplitude of a signal will be on the same scale as the amplitude of another signal. In other words, every data point collected is compared to a spectral signal of a stable source so that each signal has the same base or origin point. These calibrations can range from extremely sophisticated to relatively simple. If the drift of an instrument increases incrementally every year, a known standard can be used to factor out the drift. If multiple variables like temperature, weather, humidity, ozone concentration, sun intensity, and more are involved, there's a high possibility that a long and complicated algorithm will be needed to eliminate all of the variable effects on the detector. The calibration of the spatial aspect insures that the instrument can measure the distance or spatial distribution of the elements involved and can accurately reproduce them. There are a variety of means to accomplish this, the more popular being Modulation Transfer Function (MTF), the Contract Transfer Function (CTF), or the Optical Transfer Function (OTF). Spectral Calibration determines whether or not the instrument can detect a response based on the frequency or wavelength of the original source and allow determination of the frequency or wavelength of the data point collected[8].

Let us review some of the potential error contributions that are found in conventional spectroscopy. One of the most important error contributions is the signal to noise ratio. The signal is the measurement from the material being recorded by the detector and noise is the random and systematic errors that cause fluctuations in the signal. The signal-to-noise ratio (SNR) is merely the power of the signal compared to the power of the background noise. The higher the ratio, the more accurate and precise the data will be. In other words, the more samples or measurements taken the less impact noise will have on the results. Noise arises from a variety of different sources, which include the instrument as well as thermal, chemical, shot, and environmental noise[2]. Drift is the long term changes that occur to the instrument, detectors, sensors, and readout and must also be taken into consideration. In addition, when dealing with a source of electromagnetic radiation like the sun, interactions frequently occur that cause a non-linear response. While many non-linear responses can be observed and predicted using previous stimuli, there is always an unknown risk with radiation. The higher the intensity, the more likely that unusual reactions will be produced and recorded by the sensor. Again, this serves to decrease the precision and accuracy of the data, and the best solution is to run multiple samples[1]. The final source of error is attributed to the sensor or detector selection. Sensors are designed to measure wavelengths in a certain range; they have a high and low boundary at which they no longer recognize wavelengths at an acceptable precision or accuracy. It is useful to have overlapping sensors in terms of wavelength selection to insure that one sensor is not misreading the signal[2].

Unfortunately, as the images get more complicated and complex, moving from 1-D to 2-D or to 3-D, there are more possibilities of error. As more variables are added into an image, each variable has additional complications that could affect the signal or processing. One variable with a large impact found in 2-D and 3-D images is the spatial aspect. No longer are just qualitative and quantitative analyses occurring. Now, information concerning what the sample (pixel) is and how much there is must be decoded along with information concerning where it is in relationship to other samples (pixels). The jump in data being analyzed is astronomical compared to a 1-D, or even 2-D, image[1, 9]. When the signal is received by the instrument, it undergoes a process called quantization. This process involves converting an analog to a digital signal. Quantization with images occurs when a large continuous data stream is approximated using a smaller set of values or symbols. For imaging purposes this allows compression of the data and faster analysis time. However, care must be taken that the smaller set of values can accurately depict the larger continuous data stream. For instance, if a straight line has a beginning at A and an ending at B, the only two points needed for an accurate representation is A and B. What if the said line is actually a cosine or sine wave? In that case, points A and B would not come close to depicting the line correctly. Instead, additional points must be chosen help in the approximation of the line. How many points must be chosen, and where will the points be? This is the concept of quantization where errors frequently occur[9, 10].

Yet, another aspect of spatial imaging is non-uniform illumination of the sample. Shade, clouds, and angle of the sun will all affect the amount of electromagnetic radiation that

reaches a sample. Consider this simple problem. In the desert the ground creates a constant electromagnetic signal. A palm tree creates shade for a small patch of desert. For the shady patch of desert, even though it has the same composition as the surrounding landscape, the signal sent to the detector will appear quite different. The computer must be able to analyze, differentiate, and recognize that the signals, although different, represent the same composition. Finally, a source of error can arise from dark current, which simply put, is the continuous response of a detector despite the lack of a signal. This can be compensated for by artificially setting it to zero[1, 2, 11].

Spectral Calibration.

Spectral calibration is a method that standardizes the response of the detector(s) to the intensity of light reaching the detector and creates a mathematical formula for converting the electrical response of the detector to a more useful quantitative unit (i.e. unit of reflectance or transmission). Spectral standards, which include gray cards, color scales, and test images, are commonly used in multispectral images such as photography and video. As with multispectral images, hyperspectral images require spectral standardization. Hyperspectral imaging by definition has more variables than multispectral imaging. Multispectral imaging requires a multivariate approach to calibration due to its light intensity and multiple color characteristics. Hyperspectral imaging, with its substantially greater number of spectral features, typically requires even more variables for calibration. Under these considerations, it is clear that multispectral and hyperspectral imaging require a multivariate calibration approach[12].

Reflectance Transforms.

The first step in hyperspectral imaging calibration is changing raw data or signal intensity from the detector to reflectance values. The reflectance transformation is needed to remove the wavelength dependencies of the detector and light source. The standard transformation to reflectance is the corrected sample spectrum divided by the corrected total reflectance spectrum (Equation 2.1)[12-14].

$$R = \frac{(C - Dk)}{(REF_{100} - Dk)} \qquad 2.1$$

where R is the unitless reflectance calculated from the raw detector response of the sample spectra (C), the dark current (Dk), and the total reflectance reference spectrum (REF_{100})[12]. As Burger and Geladi point out, this is a two-point calibration based upon 0 and 100% total reflectance, but it is generally called the one-point calibration model. We will also use their nomenclature and describe this as a simple model. The simple model may be rewritten as Equation 2.2[12].

$$R = \frac{-Dk}{(REF_{100} - Dk)} + \frac{C}{(REF_{100} - Dk)} \qquad 2.2$$

Or generalized to equation 2.3[12].

$$R = b_0 + b_1 C \qquad 2.3$$

where all b terms are constants. In this case $b_0 = \frac{-Dk}{(REF_{100} - Dk)}$ and $b_1 = \frac{1}{(REF_{100} - Dk)}$. This is however a very simple transformation to reflectance values, and it does not describe non-linearity of the calibration. When calibration spectra in addition to the 0 and 100% reflectance spectra are available, a least squares regression model is more useful. Equation 2.3 would then represent a linear model where b_0 is the intercept and b_1 is the slope of the best fit line. Equation 2.4 is the quadratic model[12].

$$R = b_0 + b_1 C + b_2 C^2 \qquad 2.4$$

where b_0, b_1, and b_2 are the coefficients of the quadratic. In general, higher order equations may be fitted to the calibration spectra; however, diminished returns are quickly reached past second order calibrations. As an important note, it must be remembered that reflectance transform calibrations must be completed at each wavelength of acquisition. The entire system, especially the source and the detector, is highly wavelength dependent and requires a comprehensive calibration. The coefficients b_i in Equations 2.3 and 2.4 may then be described as a function of wavelength as vectors $\boldsymbol{b}_\mathbf{i}$[12].

Internal and External Calibration.

The terms internal and external calibration were hijacked by Paul Geladi from their original meanings. In the case of internal and external calibrations, Paul Geladi's definition will be used and is described below[15]. Although the calibration of an individual detector (pixel) is not overly complicated, it is important to remember that

there are frequently many detector elements in hyperspectral imaging. One- or two-dimensional arrays are frequently employed to reduce data collection times and to add spatial complexity to the calibration. Different methods of calibration may be applied to spatially distributed arrays of detectors. Global calibration treats a detector array as a single element. Such a transformation model is described in Equations 2.1-2.4. No spatial considerations dealing with the location of individual pixel positions are taken into account with global calibration.[12]

Pixelwise calibration deals with the facts that individual pixels may have differing wavelength-specific responses, non-uniform lighting between pixels, and different non-linear responses[12]. In addition, transferring a calibration model from one instrument to another requires a transformation matrix. This is important for the implementation of spectrometers in a large scale application such as factory production lines. In this section, we will discuss external and internal standards for both global and pixelwise calibration as well as the elimination of dead pixel contributions. In general, both external and internal standards are required for a comprehensive multivariate calibration model[12].

External calibration in hyperspectral imaging.

In hyperspectral imaging, external standards are used to correct for spatial non-uniformity of detectors and sample illumination. The detectors may spatially differ in response (linearity) and in the quantity of dark-noise. It is important to complete external calibrations to correct for long-term instrument changes such as the aging of equipment[12].

As mentioned, the global calibration model treats the entire system as a whole and thus ignores all of the spatial aspects of the imaging system. Global calibration can be thought of as a stepwise process. First, a series of external standards with varying percent reflectance values is scanned to form a series of hypercubes (one for each standard). The median spectrum from each hypercube (standard) is then used to build the global calibration model for each wavelength. The simple transform model (Equation 2.1) based upon 0 and 100% reflectance or the linear (Equation 2.3) or the quadratic (Equation 2.4) models based upon calibration sets may be used to form the calibration matrix of vectors $\mathbf{b_i}$. In the case of global calibration, there will only be one set of $\mathbf{b_i}$ vectors for each wavelength as opposed to a hypercube B_i made up of $\mathbf{b_i}$ vectors for each detector (pixel)[12].

The pixelwise calibration model, on the other hand, corrects for detector and illumination inconsistencies at each individual pixel rather than considering the median of the whole in the global calibration model. The process for creating the calibration matrix is very similar. As in global calibration, a series of external standards with varying percent reflectance values are scanned to form a series of hypercubes (one for each standard). The difference between the global and pixelwise calibration models lies in how the transformation matrix is calculated. Instead of selecting the median spectrum from each hypercube in the global calibration model, the pixelwise calibration model is computed for each pixel by using the spatially (or pixel) specific spectra from the scanned standards. It is important to remember that there will be a regression vector $\mathbf{b_i}$ for each

pixel in the pixelwise calibration model. These individual regression vectors for each pixel will form a hypercube of regression coefficients, B_i[12].

Internal calibration in hyperspectral imaging.

Internal calibration takes into account factors beyond sensor and lighting issues accounted for in external calibration. These factors include thermal drift of detectors, filters, and/or light source(s) as well as power supply fluctuations, dust buildup, and other shorter term changes in the hyperspectral imager. Frequently, the aspects of internal calibration in hyperspectral imaging are addressed as instrument standardization or calibration model transformations. In this text, we will refer to this topic as internal calibration[12].

Hyperspectral imaging produces an immense amount of data. 307,200 pixels are contained in a 480 x 640 pixel image. If a mere 1% of this image is tessellated with internal standards, 3072 spectra in each hypercube may be used for calibration. After the hyperspectral image acquisition, models for adjusting the measured spectra may be calculated from the internal standards. The internal calibration model corrects for both wavelength and reflectance inconsistencies between pixels. The process may be thought of as a normalization of all "slave" hypercubes based upon a "master" hypercube. Similarly to external standards, the normalization process may be accomplished with a simple model (Equation 2.5), a linear model (Equation 2.6), a quadratic model (Equation 2.7), or other higher order model[12].

$$R_c = \alpha R \tag{2.5}$$

$$R_c = \alpha_0 + \alpha_1 R \tag{2.6}$$

$$R_c = \alpha_0 + \alpha_1 R + \alpha_2 R^2 \tag{2.7}$$

where R_c is the corrected reflectance value from the internal calibration, the α_i are the wavelength dependent constants computed from the "master" vs. "slave" values, and R is the corrected reflectance value from the external clibration.[12] These global type correction schemes must be completed on each hypercube with the α_i vector computed for that hypercube. This method is equivalent to a Piecewise Direct Standardization (PDS) with a window width of one[12, 16, 17]. This global internal calibration method corrects for changes that occur between instruments as well as in the relatively short amount of time between sampling and external calibration. It is important to remember that all spatially dependent spectral variations were accounted for in the pixelwise external calibration. Since those variations have already been corrected in the reflectance transform, there is not any benefit for a pixelwise internal calibration. The spatial variation dependencies have already been corrected in the external calibration.

Dead pixels in hyperspectral imaging.

It is quite common in array detectors and especially in diode array detectors to have some elements with unusually high or unusually low responses. These pixels are called dead pixels. Dead pixels typically lie at the threshold of detection providing a reading near 0 or 4095 for a 12 bit detector. These dead pixels may be removed by sorting the dataset of

raw intensities, which places them at the ends of the dataset. They may then be removed from the dataset based upon their proximity to the detection maximum and minimum possible values. Alternatively, the dead pixels may remain on each end as they will have little effect upon the median value used in calibration. If the mean value is used for calibration, it is important to remove all dead pixels. Dead pixels have the potential to strongly bias the mean value since they are located at the detection limits. Dead pixels may be removed with a simple univariate thresholding technique where pixels above or below set values are not included in future calculations with the dataset[12]. Another method for removal is by setting a threshold in the distance in standard deviations from the center of the dataset. Mahalanobis distances that incorporate dispersion or QBEAST distances which incorporate dispersion and skew may be used as calculation techniques[18-20]. Five standard deviations (SDs) from the dataset center is a reasonable starting point for dead pixel elimination.

Radiation Transfer Model.

In hyperspectral imaging over long distances (remote hyperspectral imaging), effects on light passing through a medium must be taken into account. Three main processes may occur: the light may gain energy from the medium through stimulated emission, the light may lose energy to the medium through absorption, or the light may be redistributed through the medium by scattering. These factors are most easily seen in the differential form of the radiative transfer equation (RTE) (Equation 2.8)[21-25].

$$R_c = \alpha R \frac{dI_\nu}{ds} = \varepsilon_\nu - \kappa_\nu I_\nu + \iint \sigma_\nu(\Omega, \nu') I_{\nu'} d\nu' d\Omega \qquad 2.8$$

where I_ν is the spectral intensity with respect to the frequency of light ν, *ds* is the distance of light traveled, ε_ν is the coefficient of emission with respect to ν and may be a function of I_ν if scattering takes place, κ_ν is the coefficient of absorption with respect to ν, and σ_ν is the coefficient of scattering which also depends upon ν and the solid angle Ω[22, 25-28]. It is apparent in Equation 2.8 that the first term represents the gaining of energy from the medium through stimulated emission, the second term represents the loss of energy through absorption, and the final term represents the redistribution of energy through scattering. It is important to remember the radiative transfer process occurs both as light propagates to the object under scrutiny and while the light is on its way back to the detector. For a passive system on a satellite orbiting Earth, light must travel from the Sun to the target on Earth and then back through Earth's atmosphere a second time to the orbiting satellite's detector. The solution of the RTE for such a process is very complex. Ultimately, the complexity of the RTE itself leads to solutions dependent upon the application. These solutions directly depend upon the various forms of the emission, absorption, and scattering coefficients. A basic solution to the RTE may be found by simply ignoring all scattering. Integration yields Equation 2.9.

$$I_\nu(s) = I_\nu(s_0)e^{-\int_{s_0}^{s}\kappa_\nu(s)ds} + \int_{s_0}^{s}\varepsilon_\nu(s')e^{-\int_{s'}^{s}\kappa_\nu(s)ds}ds' \qquad 2.9$$

Equation 2.9 may be used in hyperspectral imaging for computationally correcting the acquired data within a hypercube for radiative emission and absorption. All radiative transfer corrections would be completed on a final hypercube that had already been corrected with the calculations from internal and external calibrations. It is important to note that significant errors may arise when using this equation upon data collected from

light which traveled through a scattering medium. An entire field of specialization revolves around solving the RTE for differing purposes, and a large sub-field revolves around solving the RTE for atmospheric effects. For more information dealing directly with hyperspectral imaging and radiative transfer consult the Journal of Quantitative Spectroscopy & Radiative Transfer.

Chemometrics

As defined by the International Chemometrics Society (ICS), "Chemometrics is the science of relating measurements made on a chemical system or process to the state of the system via application of mathematical or statistical methods." This definition encompasses an enormous range of mathematical processes including calibrations, linear and multivariate regressions, pattern recognition, and data mining, as well as signal processing and analysis. There is one primary chemometric method used throughout this work. It is the process of principal component analysis (PCA). Other chemometric methods that are used for specific applications are discussed in detail in the sections where they are applied. For more information on these and other chemometric techniques several sources are especially useful[29-35].

Principal Component Analysis (PCA).

PCA is a dimension reduction technique for multivariate data in which a new coordinate system is created. The new coordinate system is developed by taking the line with the greatest variance through the multidimensional dataset and making it the first principal axis. The second axis would be the next orthogonal line with the greatest variance, and so on. In short, each new orthogonal axis contains the most remaining variance[29, 32-34, 36].

In general, a chemical mixture of non-interacting components would have the same number of significant principal components (PCs). This is assuming the components also follow Beer's Law across the collected spectral range. Any other PCs would be indicative of other sample variations such as noise. Often times the samples are much

more complex. Chemical species in a mixture may interact, not follow Beer's Law, have overlapping spectral bands, etc. This leads to complications in using PCA as a clustering technique. In general, hyperspectral images of tissues are nearly continuous in nature. Although PCA does indicate the greatest orthogonal variances, it does not have the ability to show clustering of different populations in very continuous datasets. In continuous datasets, it also does not give graphical representations of chemical similarities within hyperspectral images. PCA is used throughout this work for reducing the dimensionality of the data as a preprocessing technique.

Molecular Factor Computing

Modern hyperspectral imaging techniques can collect terabytes of spectral information per hour. The modern computer, however, is unable to save and process all of these data as quickly as they are collected. Frequently data reduction is an early step in the information process. PCA as described previously reduces data based upon variance, and although this process is useful, it is still computationally demanding. It would be better to reduce the data in the hardware of the instrument itself instead of doing so in software. Integrated sensing and processing (ISP) is one of the most commonly cited approaches. ISP makes data collection an active process in multivariate analysis. The data acquired are not excessively large and filled with useless information but are instead an information rich smaller quantity.

ISP is especially useful with optical spectroscopy. In the past, factor techniques like principal component regression and partial least squares regression have been used for data analysis[37, 38]. These factor approaches may be applied to the incoming beam of light by weighting different wavelengths of the light. This alteration of the incoming light is frequently done with various filters with differing optical properties, and they may be applied across the UV-IR spectrum[39-42]. The downside is the difficulty in producing these application specific filters, which ultimately leads to their costly and often prohibitively costly nature. An alternative would be to use molecular filters.

Molecular absorption filters may be used to correlate the factors of mathematical spectral analysis. When they are used in this way, it is called Molecular Factor Computing

(MFC). In MFC, the transmission spectrum of a compound is used to replicate the weighting in a general mathematical processing technique such as PCA. Molecular Filters (MFs) are then designed to correlate very highly with the desired information, and then these MFs are placed in the path of the incoming light. The resulting detector response is then directly correlated with the MFs and ultimately the desired information that they were tuned to represent. The advantages are substantial. First, a single element detector may be used for detecting the response (2D array for imaging). Second, there is no slit to achieve higher resolution and so there is an optical throughput advantage. Third, there are no moving parts within the spectrometer making it much more rugged. Fourth, all light is used at once resulting in a multiplex advantage over dispersive instruments. Potential applications range from the rugged spectrometers needed in space exploration to low throughput spectrometers such as near-field optical microscopes (NSOM). It has already been applied in the NIR spectral region for the determination of ethanol concentrations in water[43, 44].

Chapter Three – Imaging Bioinformatics Using Cluster Analysis with Quantile-Quantile Plots and Synchrotron IR Microspectrometry for the Study of Abdominal Aortic Aneurysm in the ApoE -/- Mouse

Introduction

Principal Component Analysis (PCA) and Cluster Analysis (CLA) have long been used to analyze various samples. CLA is a categorization applied to many different algorithms that group data based upon similar characteristics. Lasch et al. chose specific training spectra for finding specific clusters in FTIR microspectroscopic data of thin human colon carcinoma sections[45]. Using PCA and CLA techniques, the authors separated the data into six unique clusters and then calculated Euclidian distances from cluster centers to all other points for use in creating chemical maps. Although this technique is useful for some applications, it limits analysis to predetermined spectra or *a priori* knowledge of the sample. In some cases, changes in the biological matrix causes significant differences between training and test spectra of the same compound, possibly, resulting in misinterpreted data. In addition, the high spatial resolution combined with the large quantities of data from FTIR microspectrometry result in overlapping principal component distributions (PCs) in all dimensions, making it difficult to determine cluster boundaries. An approach by Lodder et al. shows separation of subclusters in contaminated pharmaceutical tablets through examination of quantile-quantile (QQ) plots.[18] The separation of overlapping distributions was accomplished by using changes in the slope in addition to discontinuities in the QQ-plot for identifying different

subcluster populations. The centers of these subcluster populations combined with the loadings from PCA reveal the chemical meaning of each cluster.

The Quantile Bootstrap Error-Adjusted Single-sample Technique (QBEAST) is a rapid measuring technique used to determine distances in standard deviations (SDs) between a cluster and other points in a hyperspace[19]. Unlike Mahalanobis distances, QBEAST is a nonparametric distance measure that accounts for skew in addition to dispersion. QBEAST distances can be calculated from cluster centers to all other points in a data set image. These distances, when scaled and plotted as colored pixels, draw a chemical map based on the identified cluster, which becomes a library entry in the biological spectral database.

An automated combination of the Lasch chemical imaging technique using QBEAST distances with the Lodder approach for subcluster analysis of large dataset creates an algorithm called Spectral Analysis by Quantile-Quantile plotting (SAQQ). This research uses SAQQ in a variety of sensing applications, from data analysis of hyperspectral images to data from a novel Near Infrared Near-field Scanning Optical Microscope (NIR-NSOM). The new solid-state spectral imager (SSSI) for extreme environments produces data in the form of spectra or factor scores (e.g., PC scores) that may be converted into chemical maps using SAQQ. Hundreds of gigabytes of hyperspectral data collected at Brookhaven National Laboratory using infrared microscopes may be analyzed with SAQQ. SAQQ produces chemical density maps from the new NIR-NSOM, which uses molecular factor computing (MFC) to implement integrated sensing and processing.

These diverse applications all share some common characteristics. The data sets are enormous. Time constraints require extremely rapid data reduction. The ratio of "pixels-to-pupils" is heading toward infinity[46]. SAQQ enables useful chemical information to be derived from all of these data sources.

In the United States, 450,000 people die annually from the consequences of atherosclerosis, a chronic inflammatory process of the arterial vessel walls[47-49]. Research is ongoing into the biochemical pathways involved in atherosclerosis. Some chemical and structural changes during the process of atherosclerosis have been identified. There has been an increase in lipoproteins noticed[50, 51] along with their oxidation in the subendothelial matrix[50, 52, 53]. The development of lipid-laden macrophages following the recruitment of monocytes also indicate the atherosclerotic process[54, 55]. Changes in the type and numbers of smooth muscle cells are noticed[56, 57]. Progression in the atherosclerotic process is noted by the increase in extracellular lipid, necrotic core and fibrous cap development, and calcification. In addition, atherosclerotic plaques may rupture causing extensive internal bleeding. The potential for rupture has been studied extensively but has yet to be fully understood[58-61]. The most recognized vulnerable plaque is a thin-cap fibroatheroma. It is characterized by a fibrous cap which is less than 65 μm thick and which is rich in inflammatory macrophages[58, 60, 62, 63].

In normal healthy vessel walls, collagen and elastin are the primary structural components. Vascular disorders such as atherosclerosis and aneurysm are a direct result of changes in the distribution of collagen and elastin. It has been noticed that the local

concentration of collagen and elastin correlates with various properties of normal aorta, atherosclerotic plaques, and inflammatory cell infiltration[64-69]. From this correlation, it may be inferred that changes in collagen, elastin, and/or smooth muscle cells indicate vulnerability of the vessel[70-77].

Abdominal Aortic Aneurysm (AAA) is defined as a focal dilation of the abdominal aorta to 150% of its normal state[78]. In 1990 in the U.S., approximately 8700 deaths were reported from AAA[79]. Many additional deaths likely occurred due to unnoticed aneurysms that ruptured causing fatal abdominal bleeding. It is estimated that in those over the age of 60, 2% to 8% of the population have an AAA[80-82]. The percentage of the population with AAA, as well as the rupture frequency, increases with age. Only 18% of 703 patients studied with ruptured AAA reached a hospital and survived surgery[83]. This means early intervention in the disease process could have a significant impact on the incidence of complications and on patient survival. However, it has proven difficult to identify incipient aneurysms, and even more difficult to study their development at the chemical level, which might some day provide a pharmacologic intervention to prevent AAA growth or rupture.

The use of a subcutaneous infusion of angiotensin II (AngII) in apolipoprotein E-deficient (apoE -/-) mice is a common method of inducing AAA for study. However, some apoE -/- mice injected with AngII develop only wall thickening without evidence of weakening and dilation. Some mice also develop wall thickening at locations in vessels where aneurysms have never been noted[84]. These unexplained differences suggest that a

molecular distinction exists, and perhaps a drug can be found for decreasing the probability of AAA formation in high-risk subjects. It is hypothesized that vessel wall thickening is due to a relative increase in the collagen I content of the vessel wall in AAA, at the expense of collagen III and elastin.

Two-dimensional FTIR microspectrometry has been used for more than a decade to create chemical images based upon selected absorption bands or mathematical combinations of bands. Yet, differences in chemical composition within a biological matrix remain difficult to examine with traditional techniques. The intense, stable collimated light produced by the synchrotron results in higher spatial and spectroscopic resolution in a shorter amount of time than traditional globar light sources[85]. It follows that more dilute samples, thicker samples, and complex biological materials are more easily analyzed with the synchrotron source. Using image detectors, tens of thousands of spectra can be obtained per hour with synchrotron FTIR microspectroscopic methods. Therefore, to determine chemical differences between normal vessel walls and those of knockout mice with thickened vessel walls, new bioinformatic methods must be developed to analyze the large quantities of data obtained from scanning at high spatial resolutions with FTIR microspectrometry.

In this study, FTIR microspectrometric data were reduced by PCA, followed by linear regression of QQ-plots of each PC as a CLA method. The loadings, representing the chemical identity of the center of each cluster, were calculated for each cluster. Then the QBEAST metric was employed to calculate distances from the center of each cluster to

all other points in the hyperspace of the image of each microscope slide section. These distances were reordered into a 2-D array corresponding to the pixels of the original 2-D microspectroscopic sample. Thus, a chemical image based upon similarity to each library cluster was created from these distances in SDs. The chemical image is a digital staining method for microscope slides that provides contrast around features that deviate from the normal chemical distribution.

Experimental

Tissue Samples.

This study used apolipoprotein E (apoE) knockout mice ranging in age from 2 - 12 months. Infusion of angiotensin II or saline by osmotic mini-pump into the subcutaneous space of mice took place at doses ranging from 500 to 1,000 ng/kg/min for 7 - 28 days. Two mm pieces from above (ak) and from below (bk) the kidney branch of the aorta from both an experimental mouse (x) and a control mouse (c) were frozen in OCT (Optimal Cutting Temperature compound). The 2 mm pieces were sequentially sectioned from top to bottom, taking 12 consecutive 5 μm slices from above and below the kidney in each mouse, resulting in 48 total samples (x-ak-1 to 12, x-bk-1 to 12, c-ak-1 to 12, and c-bk-1 to 12). These samples were mounted on two low-e glass slides (SensIR) for 2-D FTIR reflectance spectrometry. An aneurysm that was large and grossly discernible occurred above the kidney branch in the experimental sample, while only vessel wall thickening was observed below the kidney branch. All sections above and below the kidney branch in the control sample were considered normal. A visible light image of the first section

of all four groups appears in Figure 3.1. All procedures involving animals were approved by the Animal Care and Use Committee at the University of Kentucky.

Instrumentation.

Initial mapping was completed on the Perkin-Elmer Spotlight model 300 containing a globar source and focal plane array. A globar is an electrical resistance type thermal light source that is typically made of silicon carbide. It is heated to ~1200° C to emit broadband electromagnetic radiation in the infrared spectral region. Rectangular maps were taken of each sample section in their entirety with 16 scans per spectrum from 720 cm^{-1} to 400 cm^{-1} at 8 cm^{-1} spectral resolution and 6.25 μm x 6.25 μm pixel size. Further analysis with the infrared microspectrometer at beamline U2b of the vacuum ultraviolet (VUV) storage ring of the National Synchrotron Light Source (NSLS) at Brookhaven National Laboratory (BNL) in Upton, NY employed a Nic PLAN© infrared microscope interfaced to a Nicolet Magna® 860 infrared spectrometer (Thermo Electron, Madison WI). A liquid nitrogen-cooled MCT detector with maximum signal intensity at 1250 μm was used. Schwartzchild 32x and 10x all reflecting mirror lenses were used for the objective and condenser, respectively. A remote projected image plane mask before the objective produced the apertures used for single-point spectra or raster-scan mapping via a digitally-controlled motorized microscope stage. With both spectrometers, a clear location on the infrared reflecting microscope slide ReflectIR® (SensIR, Danbury CT) was used to obtain a reflection background spectrum.

Analytical software.

Scores Analysis by Quantile-Quantile plotting (SAQQ) and all other analytical software were written in Matlab 7.0.1. (The Math Works, Inc., Natick, Mass.)(See Appendix A for written in house Matlab function). SAQQ calculates and displays an empirical Quantile-Quantile (QQ) plot from each scores column-vector of a Principal Component Analysis (PCA) and then estimates cluster boundaries from the QQ plot.

Once SAQQ produces the QQ plot, clusters are identified using linear regression. Because SAQQ plots the order statistics of the data relative to a normal cumulative distribution function, a linear segment of the QQ plot indicates multivariate normality in a projection of a group of spectral data points. Therefore, the various linear segments of a QQ plot represent various normal distributions. SAQQ uses linear regression to find these linear segments, and through the fitting SAQQ finds the multidimensional means and standard deviations of the d-variate spectral data clusters. In Figure 3.2, two clusters are separable only in the x dimension; the corresponding QQ plots produced by SAQQ for each dimension demonstrate the need to analyze each dimension independently. The starting point for finding the first cluster is defined as the left-most point on the QQ plot (x_{min}, y_{min}); this is the minimum value. Cluster boundary estimations are found using the r^2 value from linear regression of a progressively lengthening portion of the quantile-quantile plot containing n data points to isolate linear portions of the plot (Equation 3.1).

$$r^2 = \left[\frac{n\sum_{i=1}^{n}(x_i y_i) - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{\left[n\sum_{i=1}^{n}(x_i^2) - \left(\sum_{i=1}^{n} x_i \right)^2 \right] \left[n\sum_{i=1}^{n}(y_i^2) - \left(\sum_{i=1}^{n} y_i \right)^2 \right]}} \right]^2 \quad \textbf{3.1}$$

Input parameters into the SAQQ routine limit both the number of clusters (indirectly, using the minimum r^2 value) and cluster size (minimum percentage of the data used to form a cluster). The first endpoint for linear regression is defined as a beginning point (x_{begin}, y_{begin}) plus a preset percentage (default percentpts = 10) of the total number of points (totalpts) (Equation 3.2).

$$x_{end} = x_{begin} + \left(\frac{percentpts}{100} \times totalpts \right) \quad \textbf{3.2}$$

The calculated r^2 value is checked against a preset limiting r^2 value (default $optr^2 = 0.9$). If the calculated r^2 value is larger than the limiting r^2 value (Equation 3.3), then the endpoint (endpt) is decreased by a preset decrement size (default decnxend = 1) (Equation 3.4).

$$\text{IF } r^2 \geq optr^2 \quad \textbf{3.3}$$

$$\text{THEN } endpt = endpt + decnxend \quad \textbf{3.4}$$

Once the calculated r^2 drops below the limiting r^2, the loop is halted and the beginning (x_{begin}, y_{begin}) and endpoints (x_{end}, y_{end}) are reset. The beginning point of the next cluster becomes the ending point of the previous cluster plus one (x_{end+1}, y_{end+1}) (Equation 3.5)

and the endpoint again becomes the beginning point plus the preset percentage of the total number of points (Equation 3.2).

$$x_{begin} = x_{end} + 1 \qquad \textbf{3.5}$$

The process continues until the end point equals the total number of points. In the final cluster, if the total number of data points is less than the beginning point plus the preset percentage of the total number of points (Equation 3.6), then the endpoint is set equal to the total number of points (Equation 3.7).

$$\text{IF } totalpts \le x_{begin} + \left(\frac{percentpts}{100} \times totalpts \right) \qquad \textbf{3.6}$$

$$\text{THEN } x_{end} = totalpts \qquad \textbf{3.7}$$

Discontinuities in the data, however, represent a finite end of a cluster that may not be identified with the r^2 test from linear regression. SAQQ checks for such areas where high slope indicates discontinuities by forcing cluster endpoints when Equation 3.8 is satisfied.

$$y_{end} - y_{begin} \le y_{end+1} - y_{end} \qquad \textbf{3.8}$$

The condition in Equation 3.8 was added to SAQQ to find discontinuities between clusters with very similar slopes. If the slope of the first cluster is greater than the slope of the second cluster, a small discontinuity would be missed. The change in slope, however, would cause the calculated r^2 value to quickly drop below the limiting r^2 value. This change would indicate a new cluster.

QQ plotting reorganizes data into their order statistics. In SAQQ, the data are indexed so that the data may be later placed back in their original order (the image pixel order). More importantly, even though there are many overlapping clusters in each dataset, the original spectra (once reordered) and the original loadings for the PC may be used to calculate the loadings for each cluster. Using inverse principal axis transformation and the loadings, the center of each cluster is calculated.

The QBEAST metric is then employed to measure the distance between each cluster's center and all other points. QBEAST distances are similar to Mahalanobis distances in that they are distances in multidimensional standard deviations (SDs) from a point in hyperspace to a calibration set. The distances are given in terms of SDs of the calibration set. Mahalanobis distances resemble a "rubber yardstick" whose length (correlating to the parametric dispersion) depends upon the orientation in which it is directed. The QBEAST distance resembles a rubber yardstick with a nail in it, and the nail is "driven" in at the center of the calibration set. The stretch of the Mahalanobis metric is symmetric, while the stretch of the QBEAST metric can be symmetric or asymmetric, depending on the skew of the data. Therefore, the QBEAST metric is free to describe unusual cluster shapes more accurately. See Figure 3.3 for a graphical comparison of the three types of calculated distances.

Chemical images are reconstructed by placing the distances of the scores from the centers back into the image order in which they were acquired. These distances are written to a 2-D array that is defined by the pixel width and height of the acquired spectral image. This

2-D distance array is used to create chemical density images with a continuous color map correlating to the distance values by setting the shortest distance to dark blue and the longest distance to dark red, and plotting one pixel per distance value.

Data Analysis.

The first analysis (Trial 1) used a training set of 1000 samples by 1 dimension generated by Matlab to determine SAQQ's sensitivity for distinguishing between two clusters in the synthetic dataset. The training set was randomly chosen from a normal distribution with a mean of zero and a variance of one. A test set was created from an exact copy of the training set with either an added offset (in SDs of the training set) (Figure 3.4) or multiplied scaling factor (in SDs of the training set) (Figure 3.4B-C) followed by recentering the mean. (Because SAQQ examines only one PC at a time, it was not necessary to use a multidimensional data set as the training and test set.) The number of SDs required for statistically significant ($p < 0.05$) cluster separation was determined.

The second analysis (Trial 2) used a training set of 80 real samples in 40 dimensions to determine SAQQ's sensitivity for distinguishing between a real data cluster and a second cluster of a differing size or location in hyperspace. The training set was drawn from PC 1 of a rectangular 113 by 102 pixel IR spectral data set, which comprised 11526 samples by 410 wavelengths. The area (highlighted in Figure 3.5) was chosen from the inner vessel wall of x-bk-1 because of the uniform (as determined by spectrometry) chemical composition of the area. Again, the test set was created from an exact copy of the

training set with either an added offset or multiplied scaling factor. The number of SDs required for statistically significant ($P < 0.05$) cluster separation was determined.

The third analysis (Trial 3) was an example of the SAQQ routine processing the 11526 samples by 410 wavelengths (reduced to 40 dimensions by PCA) of x-bk-1 to identify clusters for which chemical maps and loadings could be calculated. The chemical maps from this example, along with all other chemical maps of the first 10 PCs, were used to identify an 11 x 9 pixel area (shown in Figure 3.6) of interest for further investigation by synchrotron FTIR microspectroscopy. SAQQ analysis was subsequently applied to that smaller region at higher spatial resolution.

Cross Validation.

SAQQ was validated using 40 spectra of endothelium selected from each of 4 adjacent normal murine abdominal aorta (160 spectra total). These spectra were concatenated into a single data set and processed as all others processed with SAQQ. Using the default SAQQ settings, each of the first 10 PC scores containing all 160 spectra was defined as a single subcluster by SAQQ. Figure 3.7 shows selected pixels of endothelium from the first section of aorta. Figure 3.8 is a quantile-quantile plot showing a single subcluster.

Results

The offset of the test set in Trial 1 by addition of multiples of the SD of the training set yielded separation of the two data sets at 3 SDs ($P = 0.05$). Figure 3.9A shows the addition of 2 to 4 SDs in 0.5 SD increments with the separations denoted by the shade of

the data points (plots are offset by scalar addition to the y-axis for clarity). Here the training set and test set are identical in all but their position in one-dimension, so reversing the training and test set would yield identical results.

The scaling of the test set by multiplication of it by multiples of the SD of the training set yielded separation of the two data sets at 3 SDs (size increase) and 0.3 SDs (size decrease). Figure 3.9B shows separation with multiplication from 2 to 4 SDs in 0.5 SD increments. Figure 3.9C shows the separation with multiplication from 0.1 to 0.5 SD in 0.1 SD increments.

The offset of the test set in Trial 2 by addition of multiples of the SD of the training set yielded separation of the two data sets at 4 SDs. Figure 3.9D shows the addition of 3 to 5 SDs in 0.5 SD increments.

The scaling of the test set by multiplication of it with multiples of the SD of the training set yielded separation of the two data sets at 2 SDs (size increase) and 0.4 SDs (size decrease). Figure 3.9E shows separation with multiplication from 1 to 3 SDs in 0.5 SD increments. Figure 3.9F shows the separation with multiplication from 0.2 to 0.7 SD in 0.1 SD increments.

SAQQ designated six overlapping multivariate normal clusters in PC 1 of the preliminary Spotlight data with an r^2 adjusted from the default to 0.94 (Figure 3.10). Chemical maps of the distances calculated from the centers of these clusters appear in Figure 3.11 with

their loadings in Figure 3.12. The area in Figure 3.6 was examined in further synchrotron analysis. SAQQ applied to the synchrotron data identified three clusters in PC 2 (Figure 3.13). PC 1 did not show more than a single cluster. Figure 3.14 shows loadings (A-C) and chemical maps (D-F) of this area. The loadings correlate very well with collagen I, collagen III, and elastin by the weak peaks at 1225 cm^{-1} and 1475 cm^{-1}, and the more intense peaks at 1550 cm^{-1} and 1675 cm^{-1}. In particular, the peak at 1740 cm^{-1} in Figure 3.14C correlates with collagen I and the lack of this peak in Figure 3.14A correlates with collagen III and elastin. There is a significant spectral contribution from water in these loadings in the 3400 cm^{-1} region.

Discussion

Because SAQQ identifies both overlapping and non-overlapping clusters in PCA scores, the detection limit of spectroscopic data processed by SAQQ is based upon the amount of overlap between clusters. In both Trial 1 and Trial 2, SAQQ was able to detect clusters that were displaced between 2 and 3 SDs. Specifically, 1 – 5% of the data from each cluster may overlap and the algorithm still identifies two distinct clusters. In addition, SAQQ detected and separated a test set that was 2 to 4 times larger or 2.5 to 5 times smaller than the training set. The dynamic range for SAQQ is between 95% and 99% and ranges from the previously defined detection limits to infinitely separated clusters of data. The linear least squares method of fitting line segments to the QQ plot in SAQQ is not as accurate of a method of fitting when compared to visual analysis. A concave or convex portion of the graph often results in too few or too many data points designated to a particular cluster (an under or overshooting of the cluster boundary). Future optimization

with approaches from both directions will increase the accuracy of SAQQ. Regardless, SAQQ is useful as an automated digital staining technique for providing contrast to regions in tissue sections where further analysis is needed as well as for separating clusters to determine statistically significant chemical differences in biological samples.

While it is easy to select an r^2 value in SAQQ that creates too many or too few clusters, it is more challenging to choose the r^2 value that will cause SAQQ to identify perfectly only the actual clusters. Because of this difficulty, a higher r^2 value was selected initially so that SAQQ identified more clusters. Visual interpretation of the QQ results was then used to select the appropriate clusters (see Trial 3). A function could be developed to choose an r^2 value from a synthetic dataset produced from parameters of the QQ-plot being analyzed. At publication of this dissertation, such a function has not yet been implemented.

Often, a bias results from curvature in the QQ-plot that causes SAQQ to continue beyond a curve and stop slightly past where a linear segment should end (Visible in the bottom plot of Figure 3.9B and Figure 3.9E). Optimization of SAQQ that included linear regression starting from both ends of the plot to correct this issue was generally successful. In rare cases, continuing beyond the curve in each direction resulted in a different number of clusters being identified. Due to these isolated incidences, linear regression in both directions has not been fully implemented at the time of publication. User bias is also introduced in the adjustment of the r^2 input value. As a numerical method of analysis with no randomness, the SAQQ program is precise. Any other

problems with precision, bias, and accuracy will result from the instrumentation, sample preparation, and data collection technique.

In the Trial 3, SAQQ identified six multivariate normal clusters in the preliminary Spotlight data at optr^2 = 0.965. Visual analysis of the linear portions of the QQ plot in Figure 3.9 suggests that only 2-4 clusters exist in PC 1 of these hyperspectral data. The loadings and chemical maps also suggest fewer clusters exist. It is possible that only clusters A and F in Figure 3.11 and Figure 3.12 are important clusters in PC 1 of this data set. In this data set, an optr^2 value of 0.96 resulted in only 4 clusters and an optr^2 value of 0.97 resulted in 8 clusters. Due to the curvature of the QQ plot, neither of the two optr^2 values resulted in clusters whose endpoints were at areas of the QQ plot where the slope changes. The most difficult challenge with SAQQ is determining the best optr^2 value for cluster determination and its resulting image recreation.

SAQQ identified three clusters in PC 2 of the synchrotron data set. The loadings in Figure 3.14B were visually noted to be somewhere between Figure 3.14A and 3.14C. Again, visual examination of the chemical maps and loadings in Figure 3.14 suggests that only clusters A/D and C/F were important.

The Nic PLAN© infrared microscope interfaced to a Nicolet Magna® 860 infrared spectrometer was in continuous operation for 4 days with samples replaced every 50 minutes. There were no instrumental failures during this time. Roughly 100 pixels by 1000 spectral elements may be collected per hour with the instrumentation on U2b.

These 100k data points may be analyzed for a specific analyte in under 5 minutes using the quantile-quantile analysis portion of SAQQ on a 3.2 GHz hyper-threaded processor with 1 GB of RAM. Imaging with SAQQ using QBEAST requires ~0.1 s per pixel. The Perkin-Elmer Spotlight was used intermittently for 10 days without failure. Roughly 25k pixels by 410 spectral elements may be collected per hour with the Spotlight. These 10 million data points may be analyzed for a specific analyte with the quantile-quantile portion of SAQQ with the above computer in under 15 minutes. Again, imaging requires ~0.1 s per pixel. In addition to the automated sample collection and processing, sample preparation time is reduced as staining is not required.

It is important to note that SAQQ is likely to be useful beyond the simple application of analysis of overlapping clusters in 2D microspectrometry of apoE -/- mice. SAQQ cluster analysis has applications as varied as studies of topographical scans of the surfaces of stars to astrobiological studies, where new planets are scanned for water deposits or the presence of compounds related to life. Multidimensional clusters may even be studied to understand the anomalies associated with hurricanes and wildfires. SAQQ has applications in virtually any field that lends itself to cluster analysis.

Conclusion

Principal Component Scores Analysis of Quantile-Quantile plots (SAQQ) is a useful as a digital staining technique for identifying significant chemical differences in 2-D FTIR microspectroscopic analysis of aortic aneurysms in apoE -/- mice. Specifically, SAQQ allows progress in determining the chemical process behind AAA formation through its

unique ability to "stain" based upon chemical significance. In particular, the use of intense broadband emission from the synchrotron allows probing to the cellular level and with near-field techniques to the organelle level. By understanding the chemical reasons behind AAA formation, improved methods of identification and disruption of their progression of formation may be found. Ultimately, many lives otherwise lost to AAA rupture will be saved.

In general, SAQQ identifies similarities and differences among data by using QQ plots and combines these with image recreation to digitally stain a recreated image of the sample. Furthermore, SAQQ allows synchrotron users to identify specific areas of interest for further analysis resulting in more efficient use of synchrotron resources. SAQQ is ultimately useful for a much broader range of problems that require a general clustering technique and image recreation for the identification of overlapping clusters.

Chapter Three Figures

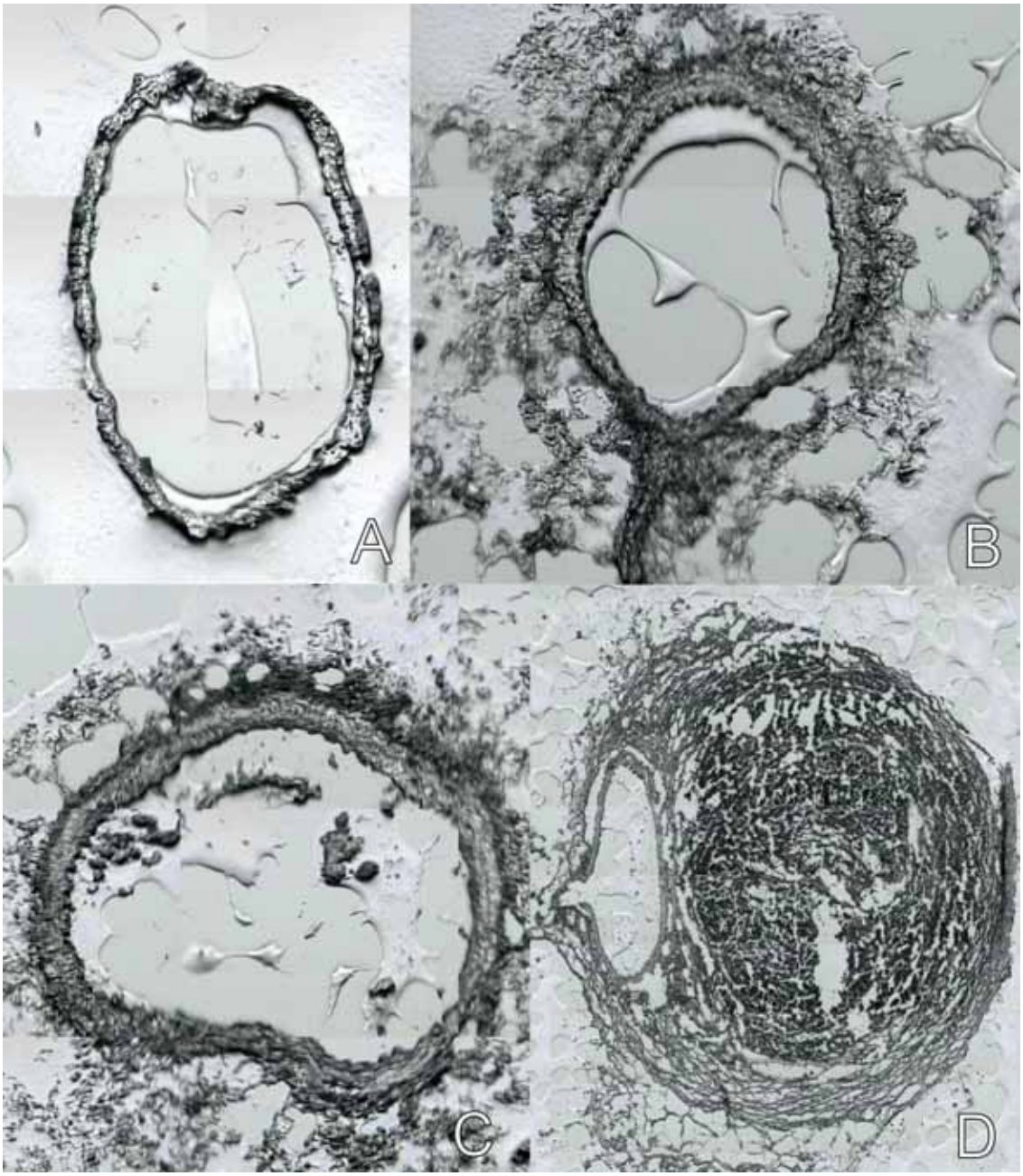

Figure 3.1: A visible light image of the first section of c-ak-1 (A), c-bk-1 (B), x-bk-1 (C), and x-ak-1 (D). The naming convention was c (control) or x (experimental) -ak (above kidney) or bk (below kidney) - section number.

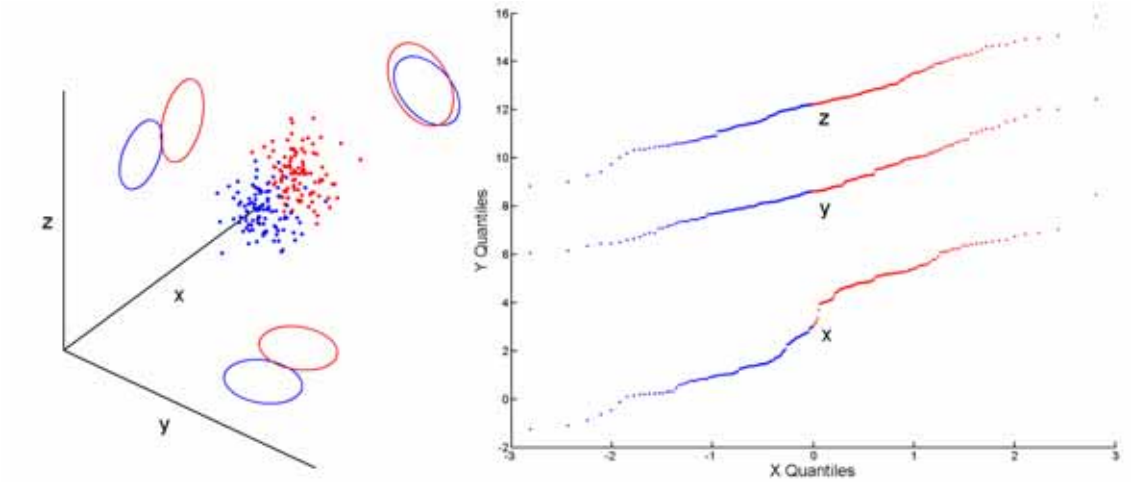

Figure 3.2: A demonstration of two clusters that is separable only in the x dimension. Left – projections (indicated by 2 STD ellipses) onto the xy, yz, and xz planes. Right – QQ plots of the clusters with the clusters indicated in each of the x, y, and z dimensions. Only the x dimension is separable using SAQQ.

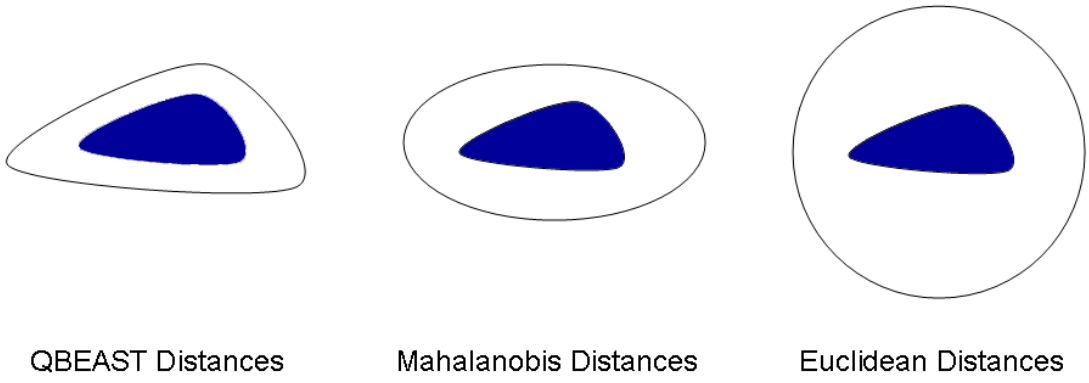

Figure 3.3: Graphical comparison between Euclidean, Mahalanobis, and QBEAST distances.

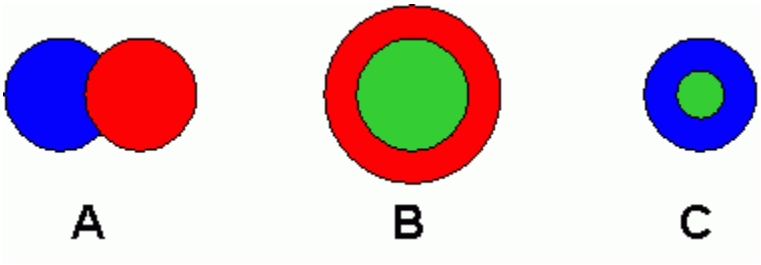

Figure 3.4: Training and test sets in hyperspace with the training set translated in one dimension (A), increased in size (B), or decreased in size (C).

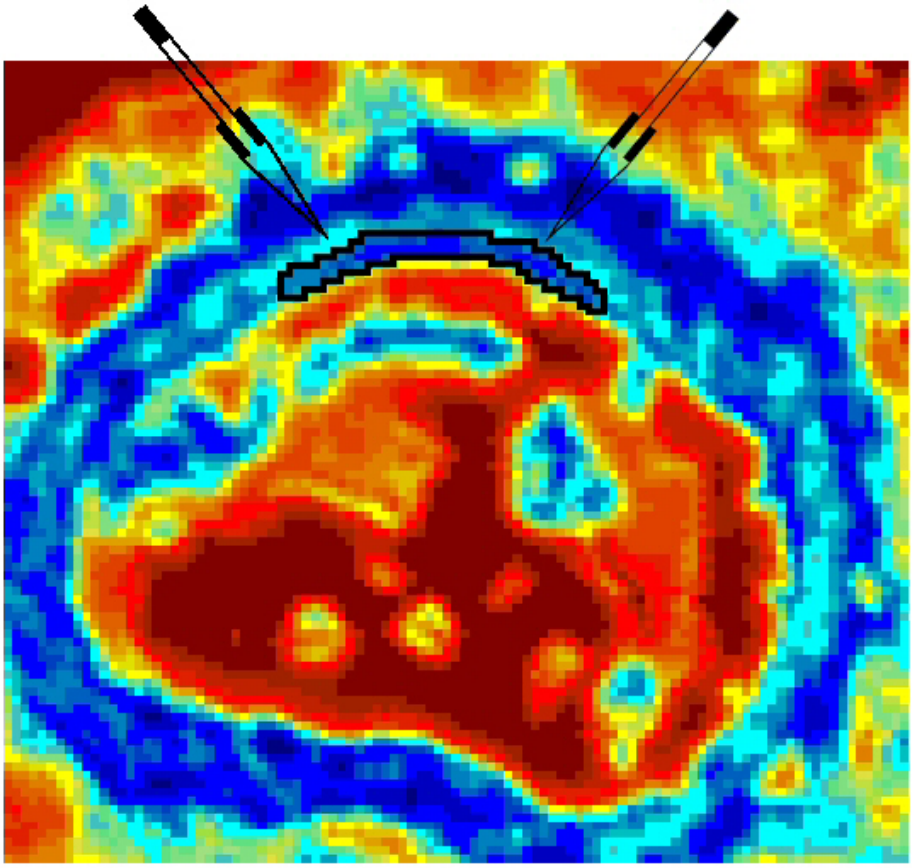

Figure 3.5: Area selected of x-bk-1 (Fig. 1-C) for the second analysis due to its relative chemical uniformity (circled in black).

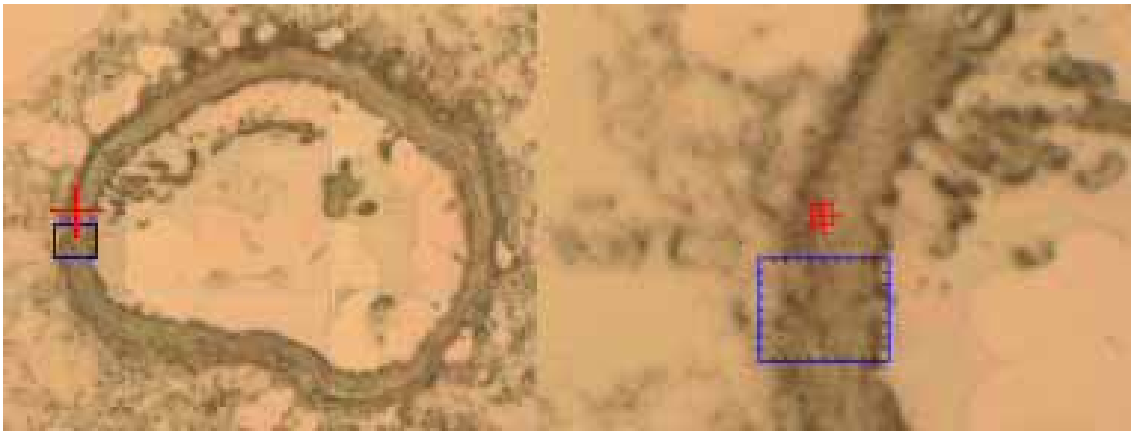

Figure 3.6: 9 x 11 μm area of x-bk-1 selected for synchrotron FTIR microspectroscopy.

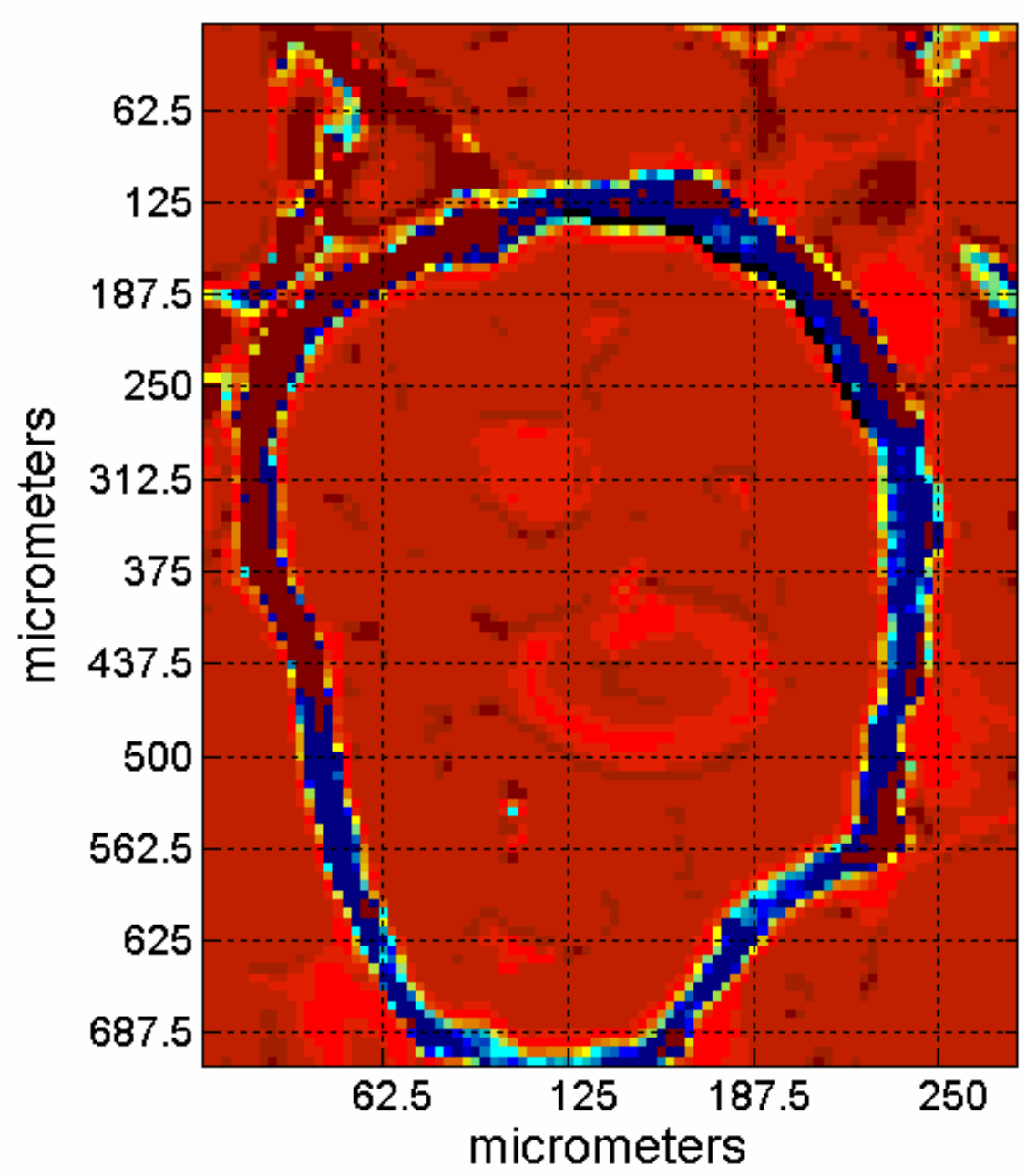

Figure 3.7: 40 pixels (denoted in black) of endothelium selected from the first murine aorta used in cross validation of SAQQ.

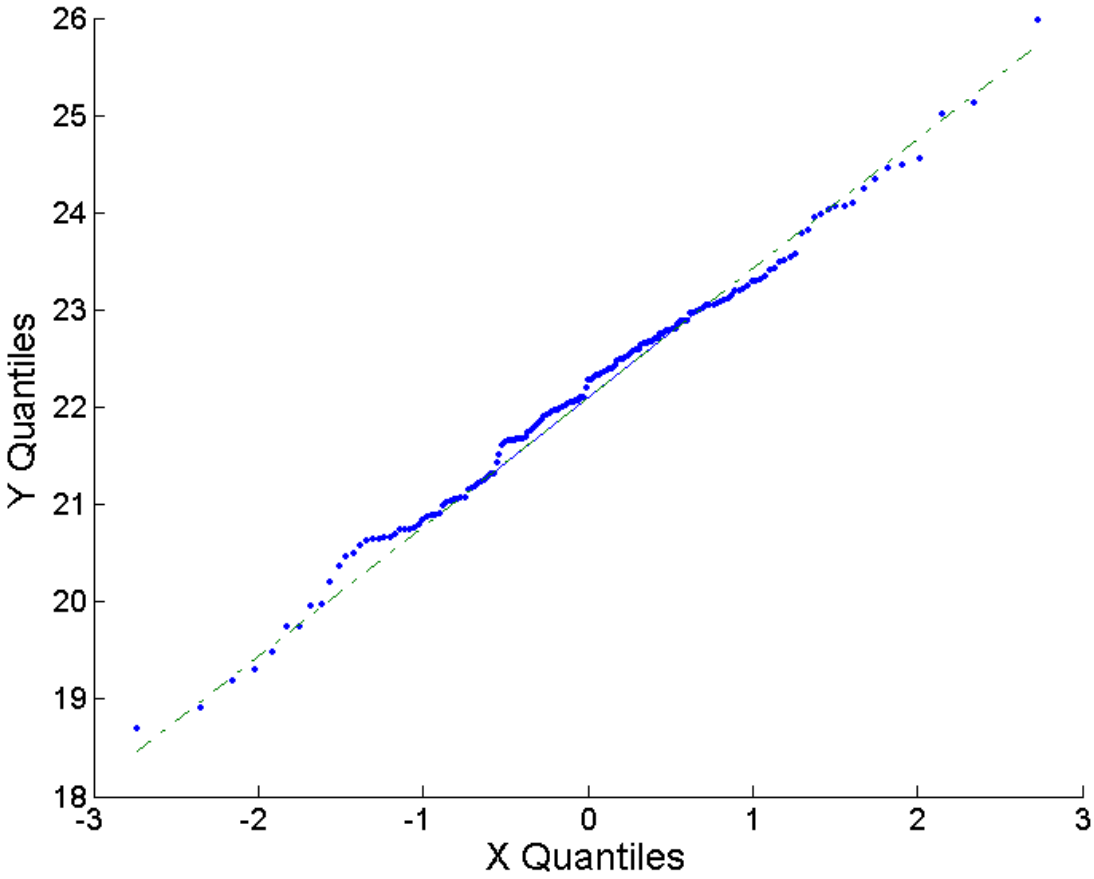

Figure 3.8: Quantile-quantile plot produced by SAQQ using default values, which shows 160 pixels (spectra) of endothelium from 4 adjacent samples of aorta falling within the same subcluster.

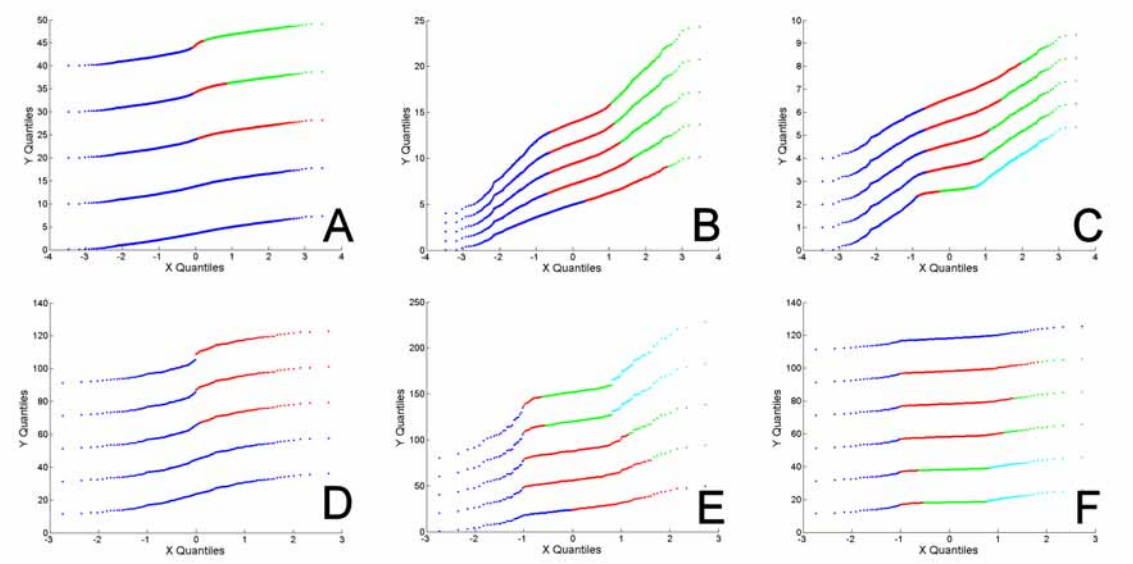

Figure 3.9: QQ plots of clustering of the first (A-C) and second (D-E) analysis. The plots in each graph are offset bottom to top for clarity. A - addition of 2 to 4 SDs in 0.5 SD increments to the test set. B - multiplication of test set scale from 2 to 4 in 0.5 increments. C - multiplication of test set scale from 0.1 to 0.5 in 0.1 increments. D - addition of 3 to 5 SDs in 0.5 SD increments to the test set. E - multiplication of test set scale from 1 to 3 in 0.5 increments. F - multiplication of test set scale from 0.2 to 0.7 in 0.1 increments.

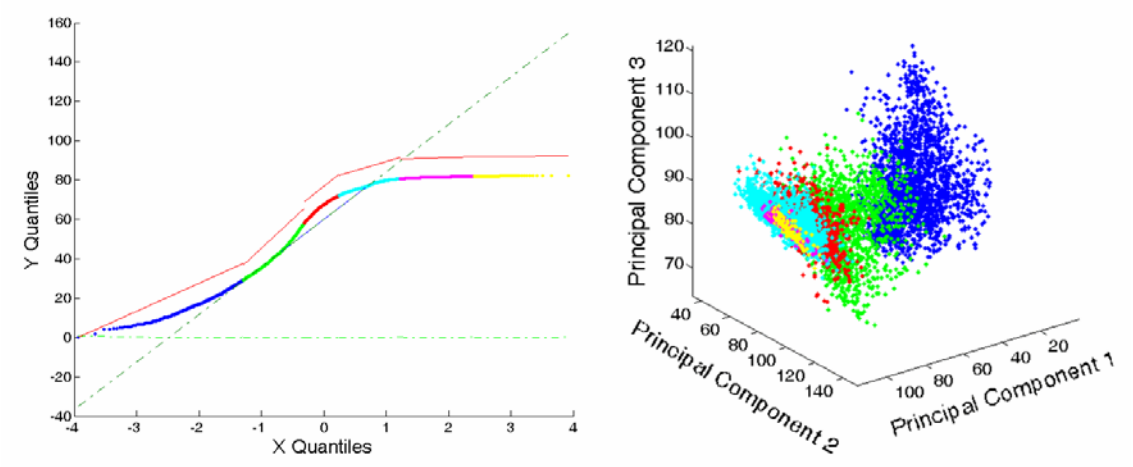

Figure 3.10: Left – QQ plot of PC1 from SAQQ analysis of x-bk-1 showing the separation of clusters. The line segments above the plot are lines of best fit offset for clarity with residuals denoted by the horizontal dotted line. The diagonal line represents a normal distribution. Right – Plot of the clusters created by SAQQ.

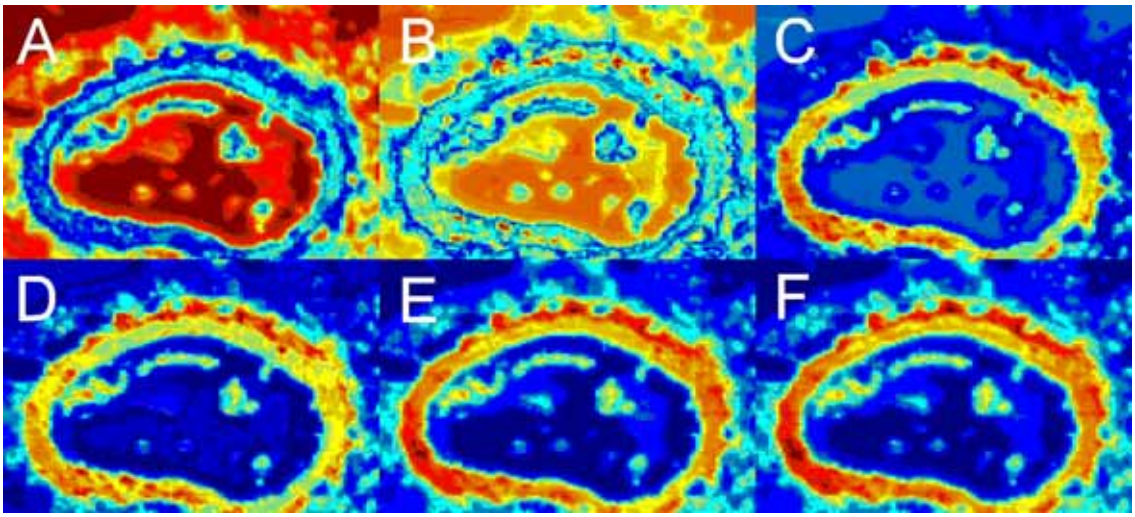

Figure 3.11: Chemical maps from PC 1 of x-bk-1 by calculating QBEAST distances from the centers of clusters identified by SAQQ. A – E identify the cluster for comparison with the loadings in Figure 3.12.

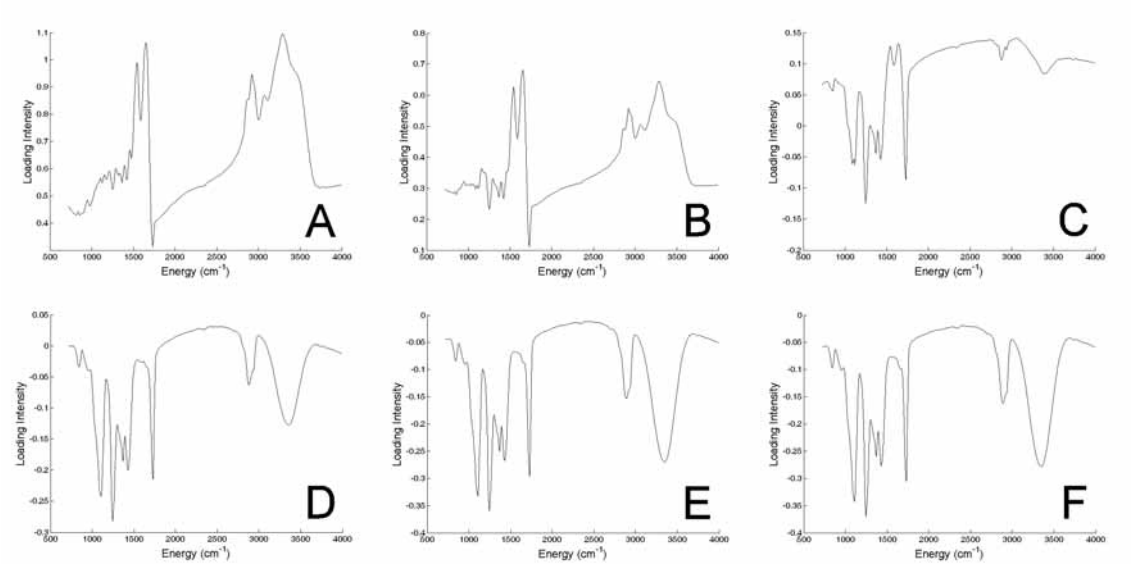

Figure 3.12: Loadings of the centers of clusters identified by SAQQ from PC 1 of x-bk-1. A – E identify the loadings that correspond to the chemical maps in Figure 3.11.

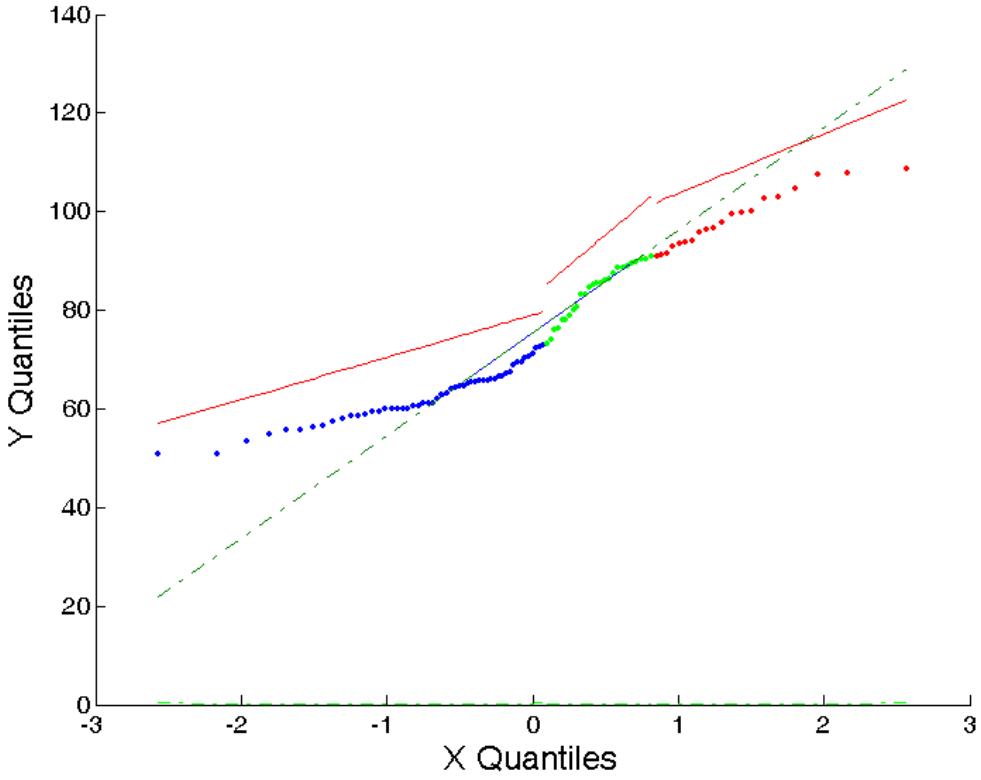

Figure 3.13: Three clusters in the QQ plot of PC 2 from SAQQ analysis of the 9 x 11 pixel area of x-bk-1 identified in Figure 3.6. Color shows separation of clusters on the QQ plot. The red line segments above the plot are lines of best fit offset for clarity with residuals denoted by the horizontal dotted green line. The diagonal line represents a normal distribution.

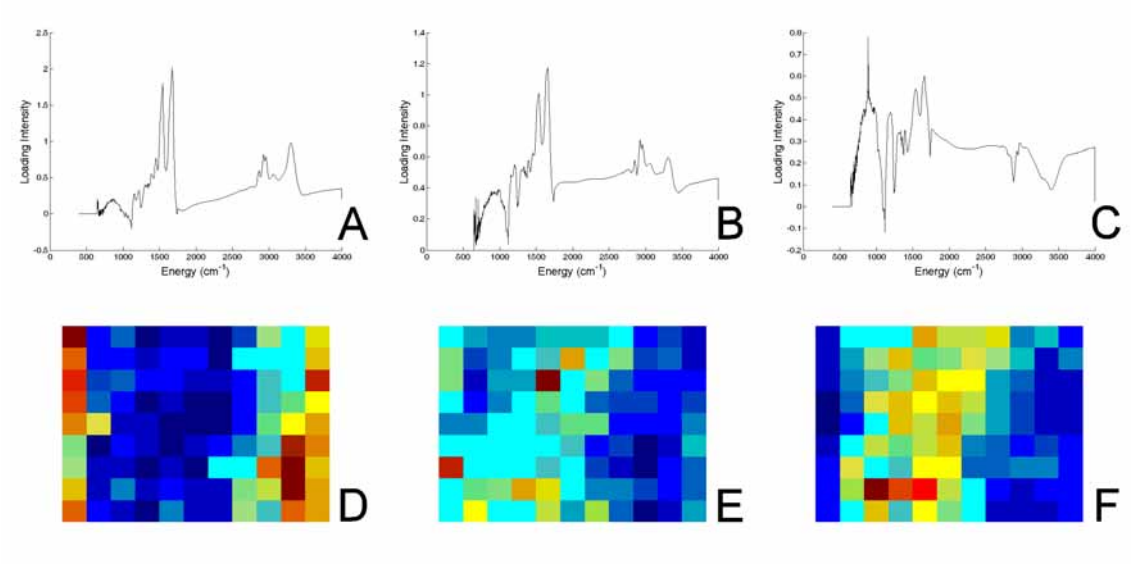

Figure 3.14: Loadings (A-C) and chemical maps (D-E) from the clusters from PC 2 formed by SAQQ analysis of the 9 x 11 pixel area of x-bk-1 identified in Figure 3.6.

Chapter Four – Application of Spectral Analysis by Quantile-Quantile Plots to Data Collected from a Novel Solid State Spectral Imager (SSSI)

Introduction

The first spectrometers contained prisms rotated for wavelength selection, and spectrometers rapidly developed into the precision filter, grating, and interferometer instruments that are now used for wavelength selection. Unfortunately, these wavelength selectors are sensitive to shock and extreme acceleration, leading to their limited field application. Additionally, the sources and multi-element detectors in spectrometers have relatively narrow spectral bands.

A Solid State Spectral Imager has capabilities beyond those of a standard spectrometer. First, the SSSI has no lenses, no focusing optics, no moving parts, and only a single-element detector. The SSSI uses Hadamard, CRISP (Complementary Randomized Integrated Sensing and Processing) or SCRISP (Shortened CRISP) encoding or pulsed frequency modulation on an expandable 5 x 5 multiwavelength light-emitting diode array for illumination of an analyte. The resulting reflectance spectrum is collected with one detector. An inverse Hadamard transform (HT) or an FFT can be calculated on the resulting data at the detector to achieve a spectral intensity value for each pixel in an image. With CRISP and SCRISP encoding, the detector voltage is directly proportional to analyte concentration. Principal Component Analysis (PCA) of the spectra from HT or FFT processed detector signals yields subnanometer spectral resolution and 5 mm spatial resolution at 10 m. Data collection and processing can be completed in less than 300 ms.

Hadamard encoding is based on the Hadamard Transform (Equation 4.1) where all rows are mutually orthogonal[40, 86-88].

$$X(b) = \left(\frac{1}{2}\right)^{\frac{n}{2}} \cdot \sum_{a=0}^{N-1} x(a)(-1)^{\sum_{k=0}^{n-1} a(k)b(k)} \qquad \textbf{4.1}$$

The Hadamard transform has sequences with a nearly equivalent number of 1 and 0 (on and off) states. This leads to equivalent usage of all light sources. The covariance between all sequences is equal and decreases as the number of diodes and pulse sequence length increases. In addition, Hadamard encoding allows for multiple diode illumination of samples rather than illuminating samples with alternating single diodes (i.e., the HT provides the multiplex advantage). An example of how three wavelengths are demultiplexed is depicted in Figure 4.1.

As an inexpensive and rugged remote sensing instrument with high spatial and spectral resolution at long distances, this Solid State Spectral Imager is useful in diverse environments. Applications of this instrument range from *in vivo* fiber optic studies of biofilm on teeth to fast analysis of pharmaceutical dosage formulations and spectral imaging in the vicinity of a Mars rover.

Application of SAQQ to the data produced by the SSSI highlights anomalies in an area surveyed as well as recreates chemical images of the area. Chemical density maps based upon distinguishing spectral characteristics can be created for visual analysis.

Deployment to Mars on a rover would allow the onboard computer to process the data collected at night from a SSSI attached to a mast above the rover. During the next day, the rover could move to the site of anomalies and conduct more detailed tests for various substances. The calculated loadings from subcluster populations may be compared to a database for prioritizing the sampling areas. A more educated guess would therefore be used in sampling rather than the current method of testing areas that simply look interesting.

SSSIs are potentially disposable spectrometers with remote sensing abilities for areas where extreme shock or complete instrument loss is likely. The visible-light / NIR prototype of this spectrometer cost only a few hundred dollars. Emission wavelength of the laser diodes is the primary system cost driver. The low cost could allow SSSIs and their spectroscopic measurements to even become an integral part of college and high school chemistry labs.

Experimental

SSSI-I.

The first prototype, a 2 x 2 array of LEDs, was built to demonstrate the abilities of such a system. The SSSI prototype was built with standard electronics components, which were purchased from Digi-Key Electronics. The SSSI circuitry (schematic in Figure 4.2) allows for four independently flashed LEDs. Components were soldered to a perforated board with single-hole pads. Connections were made between components with insulated solid copper wire (Figure 4.3). 8-pin dip sockets were used for most components as

resistances and capacitances were sometimes altered for optimization of the prototype (Figure 4.4). Each LED was oscillated, with a 555 timer chip, at a differing frequency (50 to 300 Hz) based upon the resistor labeled in Figure 4.2. The two by two LED matrix was formed with two red and two green LEDs. The detection was completed with an NPN phototransistor, a 100 k ohm potentiometer and a Sound Blaster external audio sound card. A red LED was later added to this circuit to shunt the DC current to ground and thus allowing detection of only AC current. The 2x2 LED matrix, complete with detector circuit, weighed 20 - 25 g depending upon the length of leads used. It ran off of 5 - 15 VDC potential and roughly 40 mA of current depending upon the LEDs and the flash rate. All data were recorded at 44100 Hz for 10 s with MatLab 7.0.1. An FFT of the data was completed for every 0.1 s of data collection time. Models of the test grid were recreated by plotting the most intense peak intensity with respect to FFT number. For the test grids, one 10 s scan (correlating to 100 FFTs) of each line was used to create each row of the test grid.

Three tests were completed. The first was a black and white grid of squares in the pattern shown in Figure 4.5. The grid was made of 25 mm squares that were printed with a standard inkjet printer on plain white paper. This test was conducted with the LEDs 25 mm away from the paper. The LEDs and sensor unit were scanned over the test grid under indoor lighting with ambient lighting from the outdoors.

The second was a test grid of green and black biofilms together upon the top of a level concrete wall. The biofilms were abraded with a blunt piece of steel to form a grid like pattern (Figure 4.6). Scans were completed as before except under cloudy skies.

The third was a test grid of green biofilm on top of a level red brick ledge. The grid and scans were made as in the second test (Figure 4.7).

To demonstrate the usefulness of the SSSI-I a Gloeocapsa biofilm on limestone was scanned three times in the NIR (FLEX (Bran and Luebbe, Elmsford, NY) spectrophotometer) and at three visible wavelengths (467 nm, 510 nm, and 650 nm) before and after baking one and two times at 160° C for three hours.

In addition to these tests, spatial resolution was determined for remote spectroscopy using the ten-meter test. The ten-meter test employed a commercial off-the-shelf (COTS) red laser pointer (3 V, 25 mA) modulated with one 555 timer circuit as described above. The laser was scanned across a black-white-black-white-black square grid with 25 mm squares (Figure 4.8). The grid was placed 10 m away from the laser and the detector was placed 10 m away from the grid near to the laser source. The light scattered from the target was collected with a 3-inch lens. An operational amplifier circuit (Figure 4.9) was placed with its phototransistor at the focal point of the lens and the output of the amplifier circuit was sent directly to a Sound Blaster external sound card and collected with MatLab 7.0.1. The scan was completed in a darkened room. An FFT was done every 0.1 s and the peak correlating to the modulated laser was plotted.

As a final test, a magnetic tape eraser was used on the unshielded SSSI-I to apply electromagnetic radiation during data collection. This test was to determine whether high-intensity stray electromagnetic radiation would interfere with SSSI-I data collection.

SSSI-II.

The second SSSI prototype consisted of a 5 x 5 matrix of super-bright light emitting diodes (LEDs)(five diodes each of 635 nm, 609 nm, 592 nm, 522 nm, and 467 nm)(Figure 4.10) contained within a collimating cover (Figures 4.11-4.12). The diodes were controlled by a Silicon Labs C8051F120 controller and the associated circuitry built in-house (Figure 4.13– 4.17). Five colors of diodes in the visible spectrum were chosen for the prototype: red, orange, yellow, green, and blue. These diodes may be replaced with any diode or laser diode from UV - IR for broader spectral analysis. The controller was connected to a laptop computer running MatLab 7.0.1 via a serial interface. Hadamard encoded pulse sequences and detection calculations were programmed into the controller while reverse Hadamard transforms, normalization, and plotting were completed in MatLab.

Data collection is initiated by a signal sent to the serial interface of the SSSI (See Appendix B for complete SSSI-II Manual). All data are collected by the SSSI using a phototransistor connected to an operational amplifier circuit (Figure 4.9). The analog signal is converted with an on-board 0.5-5 V analog to digital converter at 12 bits. Each scan consists of 256 data points collected in both the on and off states of 25 Hadamard

encoded light sequences. The result is 50 total states with 12800 data points collected for each scan. The corresponding values of the on and off states for each Hadamard coded light sequence are subtracted to remove ambient light from the data. After subtraction, the resulting 256 data points from each of the 25 Hadamard coded light sequences are then averaged to obtain 25 16-bit equivalent intensity values. This oversampling and averaging at a rate well above the Nyquist limit provides the additional 4 bits of resolution. The final 25 16-bit resulting values are exported to Matlab via the serial connection to a graphical user interface where data undergoes a reverse Hadamard transform to obtain intensity values for each of the 25 diodes. A single scan with MatLab processing takes less than 300 ms. The switching speed of our transistors within the SSSI is sufficiently slow that this prototype requires a 5 μs delay before each datum reading for signal stabilization after the lights have switched states. These times will be significantly speeded up if we move to a commercial quality device[89].

Spatial resolution was determined by scanning across parallel black-white-black bars and determining when the response of the white bar was less than 50% of a solid white test pattern (Figure 4.18).

Spectral resolution equivalency was determined by scanning each of six colored squares 100 times (Figure 4.19). The squares were red, blue, yellow, green, orange, and white. The resulting 600 spectra of 25 data points (one from each diode) underwent PCA. Here the term “spectral resolution” is used as the ability to distinguish two different samples

that differ in "color" (wavelength). This is different from the classical definition of distinguishing between two closely spaced peaks in the spectrum of a single substance.

As a final test, SSSI-II with the 25 visible diodes (five diodes each of 635 nm, 609 nm, 592 nm, 522 nm, and 467 nm) but no collimating cover was used to collect 100 scan spectroscopic samples underwater at 3 locations around the Great Salt Lake Area (Table 4.1). Two of the samples were in cold water (near 0° C) while the Warm Springs reading was in warm clear sulfurous water (35° C approximated by touch) with the detector in a very low sensitivity state due to the bright sunlight from clear skies. Absorbance measurements were collected in shaded areas with 30 cm pathlengths. The sample collection process may be seen in Figures 4.20-4.21.

Results and Discussion

SSSI-I.

The first experiment on the black and white test grid in Figure 4.5 was analyzed with the SSSI-I and the results are shown in Figure 4.22. This figure shows the 3-D graph with peak placement (proportional to frequency) on one axis, FFT number on the second axis, and peak height on the third. It is important to note that the black areas of the template correlate very closely with the low response areas of the graph (response increases moving down the graph). Ambient light has a very minimal effect upon the SSSI-I as can be noticed in the lack of fluctuations recorded from the dark squares of the template. Ultimately, this experiment was completed for a qualitative example of how a SSSI-I could function and was completed under the most basic circumstances.

The results to the second and third experiments with the SSSI-I on the biofilm test grids in Figure 4.6 and Figure 4.7 are shown in Figure 4.23 and Figure 4.24 respectively. The ability to distinguish between supporting materials and the biofilm was apparent in these examples.

These tests were designed to show that the concept of using LEDs oscillating at different frequencies combined with a single element detector may be used for remote spectroscopy. Laser diodes could be oscillated instead of LEDs (used in the 10 m test) and various colors/wavelengths of LEDs may be used. When each LED/Laser diode is oscillated at a different audible frequency, a simple setup with an operational amplifier and sound card may be used to collect an interferogram. Computing the FFT and using either peak height or peak area allows reconstruction of the spectral responses. The addition of more than four pixels acquired at once will result in a greater sensitivity and increased signal to noise ratio. In addition, the LEDs that were used had much lower intensity than expected, but nonetheless were still useful for acquiring data during daylight hours. Overall, it can be expected that later prototypes will have much higher signal to noise ratios due to the multiplex advantage, as well as greater wavelength selectivity, diminished pixel size, and lowered mass.

The averaged spectra for the Gloeocapsa biofilm on limestone rock are shown in Figure 4.25. The difference between the natural and baked states is most extensive at 1400 nm and 1950 nm and could easily be discerned with a NIR-SSSI-I and diodes at those two

frequencies. The results of the visible analysis of the Gloeocapsa film with three diodes are shown in Figure 4.26. It is apparent that three visible diodes are sufficient to differentiate natural and baked Gloeocapsa biofilms. In addition, the transition between living and dead Gloeocapsa biofilms could be monitored with a SSSI-I.

The results of the 10 m test may be seen in Figures 4.27-4.28. The grid was recreated almost identically. It may be noticed that there is an uneven recreation of the grid based entirely upon the difficulty in achieving an even scan rate at 10 m. Spatial resolution based upon this experiment was calculated to be 3 mm at 10 m.

The electromagnetic radiation test yielded no notable noise/response from any of the components of the SSSI-I. This is a significant find, as later prototypes could ultimately be used in environments (such as space) with high levels of electromagnetic radiation.

The usefulness of the SSSI-I can only be realized by the combination of all aspects of the SSSI-I tested here. The final instrument equipped with laser diodes of differing wavelengths (colors) oscillated at differing frequency would yield an inexpensive instrument for remote spectral analysis in extreme environments. For example, an instrument of this type could be placed on a Mars rover to scan during the night for likely indications of life. The next day, the rover could use this information to make educated guesses at where the most useful samples could be collected. The rover could then collect those samples. The SSSI-I has very low power and weight requirements which

would make it suitable for such an application. The SSSI-II expands upon this design to create a more functional SSSI for extreme environments.

SSSI-II.

The results of the spatial resolution test may be seen in Figure 4.29. The spatial resolution was determined to be 5 mm, one fourth of the spot size, which will allow high spatial resolution once laser diodes replace the conventional diodes. It is important to note that the limiting factor of the spatial resolution is the actual spot size on the target.

The plot of the first three principal components from the PCA of the dataset acquired from the equivalent spectral resolution test may be seen in Figure 4.30. The clusters with 15 STD spheres were also plotted to indicate the degree of separation (Figure 4.31). Cluster standard deviations calculated in 25 dimensions gave greater than 700 STDs between the red and orange cluster centers and greater than 1000 STDs separated between all other combinations. Based upon 2 STD spacing, these results allow calculated spectral resolution to less than 0.5 nm in the visible range with only the five colored LEDs. Again, the term "spectral resolution" is used as the ability to distinguish two different samples that differ in "color" (wavelength) by less than a nanometer.

The *in situ* water samples taken in the Salt Lake City area underwent PCA and a plot of the first 3 PCs is shown in Figure 4.32. The obvious separation in the first PC is likely due from experimental errors such as poor alignment, differences in overall absorption of light, major differences in the absorption of individual color, etc. This separation in PC1

and lack of separation in PC2 and PC3 may be seen more easily in Figure 4.33, which incorporates 15 STD ellipses surrounding the data points. The PCs other than PC1, however, contain much lower percentages of variance and separation within these PCs is not seen with PCA. SAQQ, on the other hand, indicates subclusters within the data that otherwise may not have been found. The QQ plot of the data showing three new subclusters may be seen Figure 4.34. The similar slopes of two outside linear segments in the QQ plot suggest that there are two subclusters, one larger and one smaller than the other like those in Figures 3.4B-C, 3.9B-C & D-E. Although there is not enough information to determine conclusively the significance of these two subclusters, one possibility may be the temperature difference which would give rise to different living organisms in warm versus cold water. Two spectroscopic datasets were acquired in approximately 5° C water while the third was in approximately 35° C water. These correlate nicely with the SAQQ analysis and subcluster separations of PC3. A plot of the clusters found in PC3 by SAQQ may be found in Figure 4.35. The PCA plot shows the lack of obvious separation points in the lower variance data. Without SAQQ, these variations would have been very likely missed.

Conclusions

A new era of spectroscopy has begun with remote spectroscopy and solid state spectroscopy. The SSSI is a significant example of how spectroscopy may be taken out of the laboratory and placed in extreme and diverse environments for the acquisition of spectral information. This instrument has the potential of scanning from the UV through to the NIR, IR, and even THz with the appropriate laser diodes and detectors installed. It

can operate in high electromagnetic radiation areas and can withstand extreme accelerations. These properties make it a prime candidate as a Mars rover instrument. Its high spectral and spatial resolution at 10 m also allows it to be useful in identifying various chemical compounds *in situ.*

The SSSI is a very inexpensive instrument with a total cost under $1000. SSSI's ability to rapidly analyze a sample for basic spectroscopic properties as well as being inexpensive makes SSSI a "disposable" instrument when compared with other modern spectrometers. SSSI also has no moving parts, no lenses, no focusing optics, and nearly infinite depth of field. Most spectrometers are far too costly and not rugged enough to be used in applications where eventual destruction is a probability. This spectrometer, however, may be placed in volcanoes before an eruption to monitor various gaseous emissions or changes in the soil chemistry. SSSI could be used on rovers in mine fields or in buildings where explosives are suspected. Their demise would not be as costly as other spectroscopic instruments. SSSI's low-cost and rugged attributes make it an attractive option for every high-school chemistry laboratory; it would allow for hands on understanding of basic spectroscopic principles.

Combining SAQQ with SSSI allows for more information to be gleaned from the principal component analysis. Significantly overlapping subclusters within the data may be found with SAQQ. Although it was not demonstrated with the SSSI, SAQQ may be used to recreate the images acquired with SSSI when collecting samples with many pixels. These images, like those produced in Chapter Two of AAA's may be used for

much more detailed analysis than has been previously possible. SAQQ combined with SSSI has the potential of revolutionizing classic spectroscopy.

Chapter Four Tables

City Creek Canyon Nature Preserve Exit Water	Hensley/Salt Lake Cutoff - Warm Spring Park	Great Salt Lake Marina
N 40° 47.407'	N 40° 47.413'	N 44° 44.164'
W 111 52.711'	W 111 54.086'	W 112 12.722'
~5° C	~35° C	~5° C
Figure 4.20		Figure 4.21

Table 4.1: SSSI-II sampling locations in the Great Salt Lake Area.

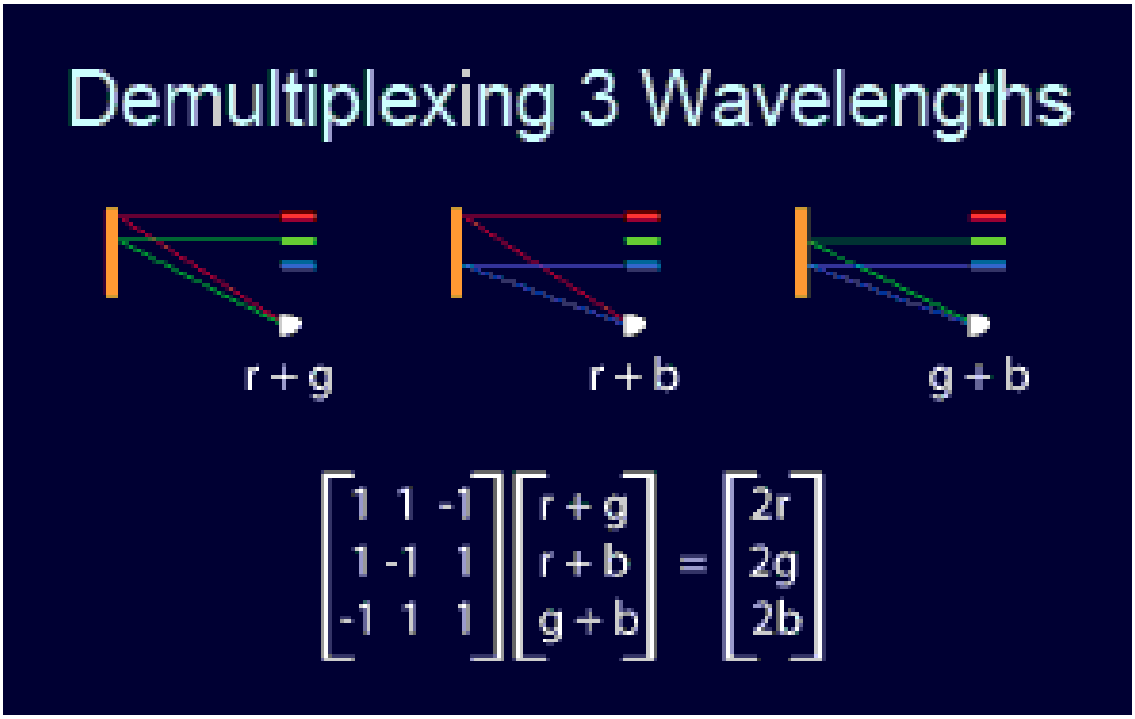

Figure 4.1: An example of how the Hadamard transform demultiplexes data from a sample. Two of the three laser diodes are active at any given instant.

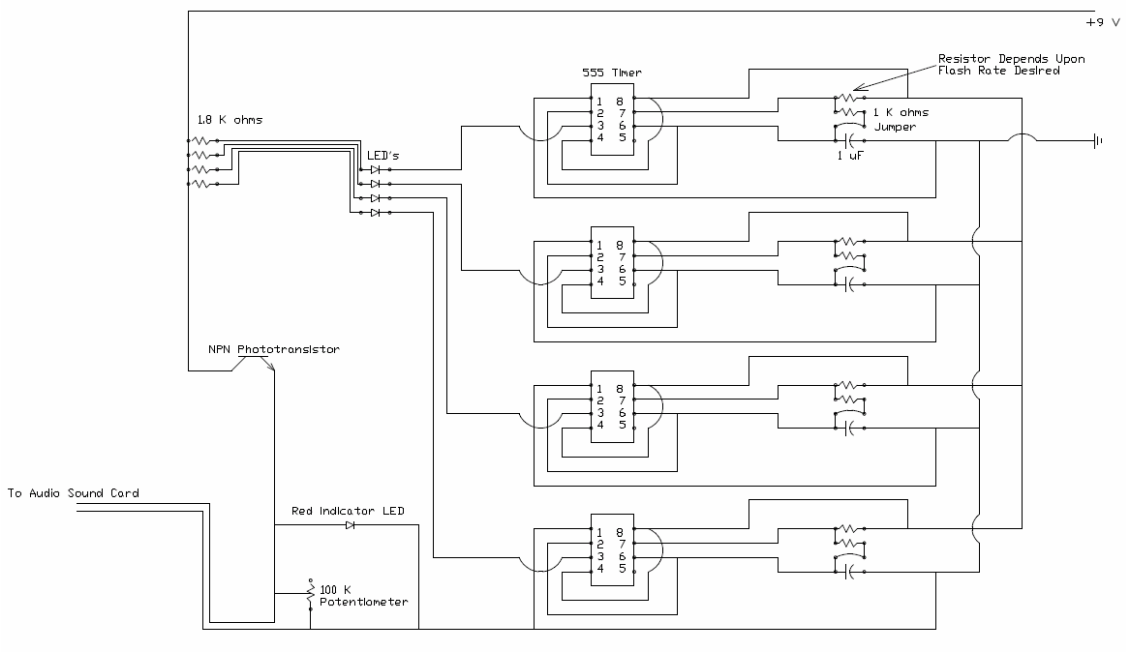

Figure 4.2: Circuit diagram for SSSI-I which used four independent 555 timer chips to flash four independent diodes at different rates and an operational amplifier circuit to amplify the NPN Phototransistor.

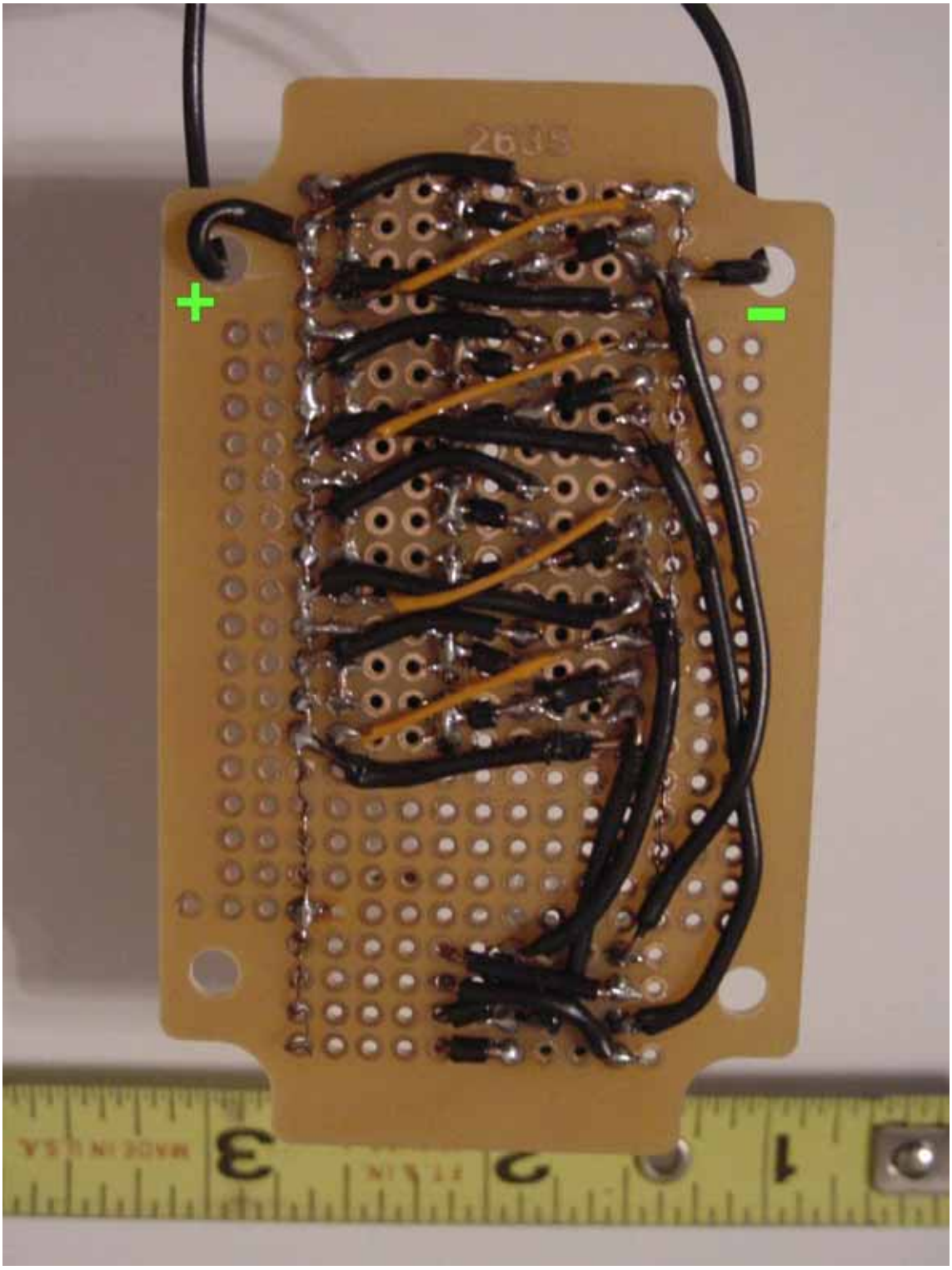

Figure 4.3: Bottom of the SSSI-I prototype.

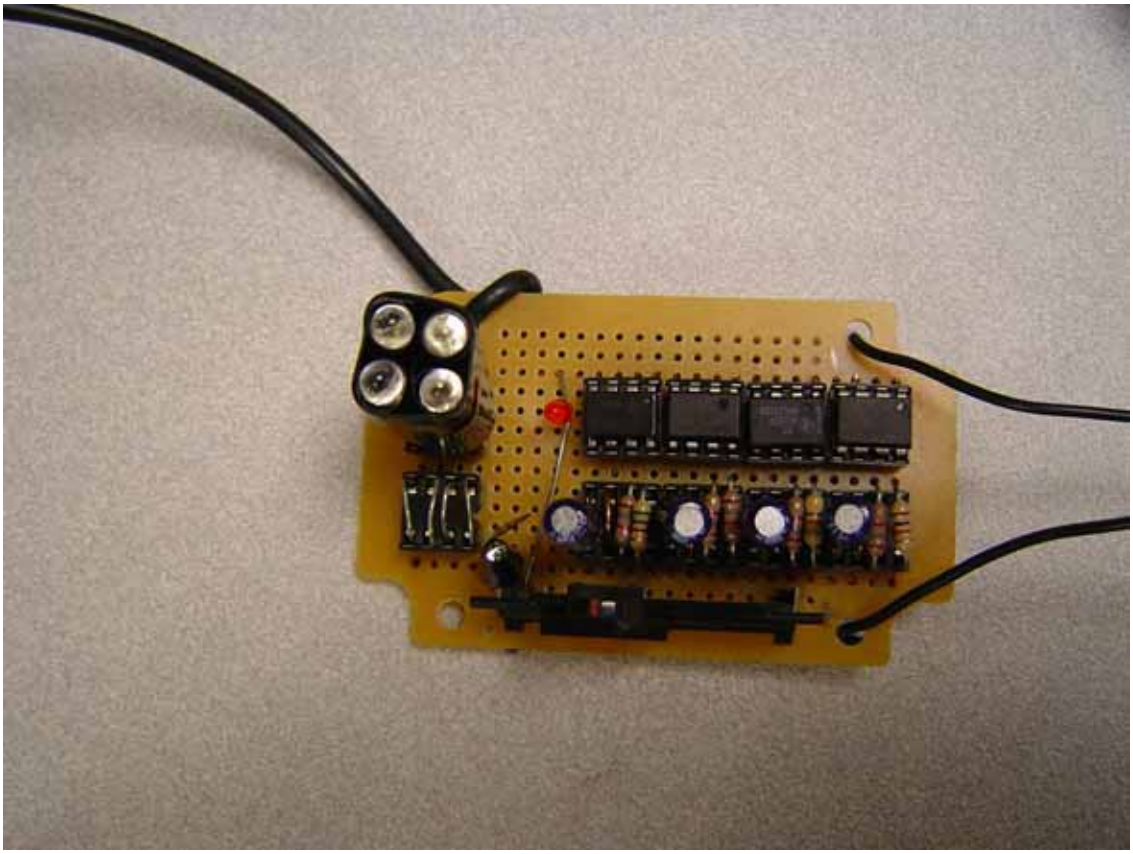

Figure 4.4: Top of the SSSI-I prototype.

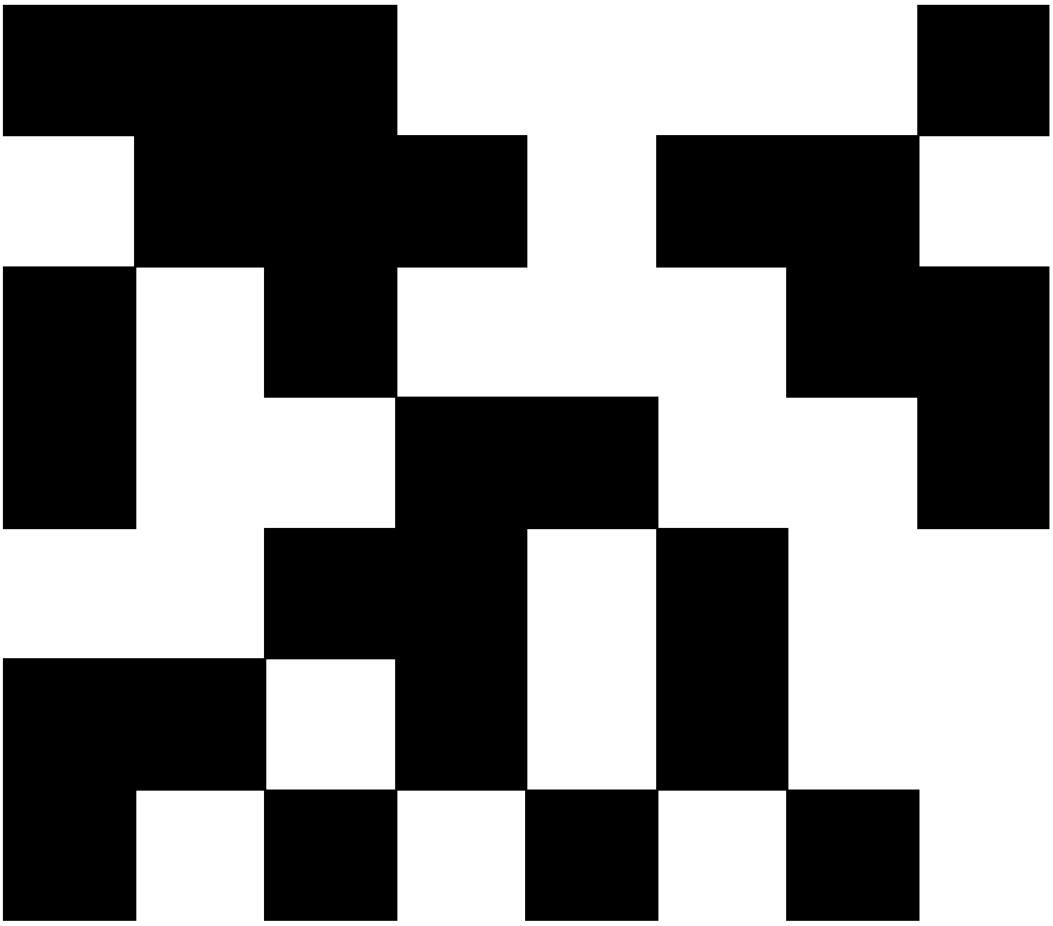

Figure 4.5: Test grid for testing SSSI-I function.

Figure 4.6: Test grid etched into green and black biofilms coexisting upon the top of a level concrete wall.

Figure 4.7: Test grid etched into a green biofilm on top of a level red brick ledge.

Figure 4.8: Ten-meter test target.

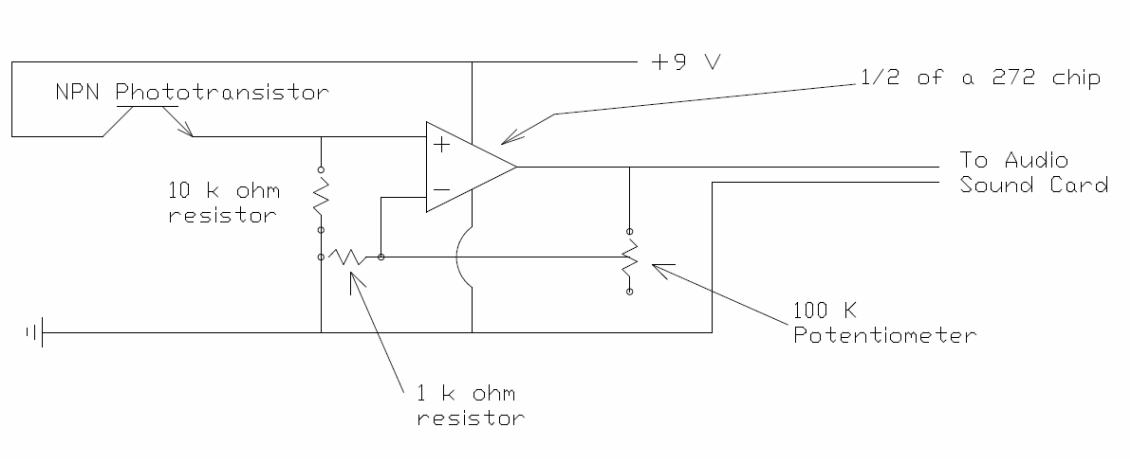

Figure 4.9: Circuit diagram for the operational amplifier circuit to amplify the NPN phototransistor on the SSSI-I and SSSI-II.

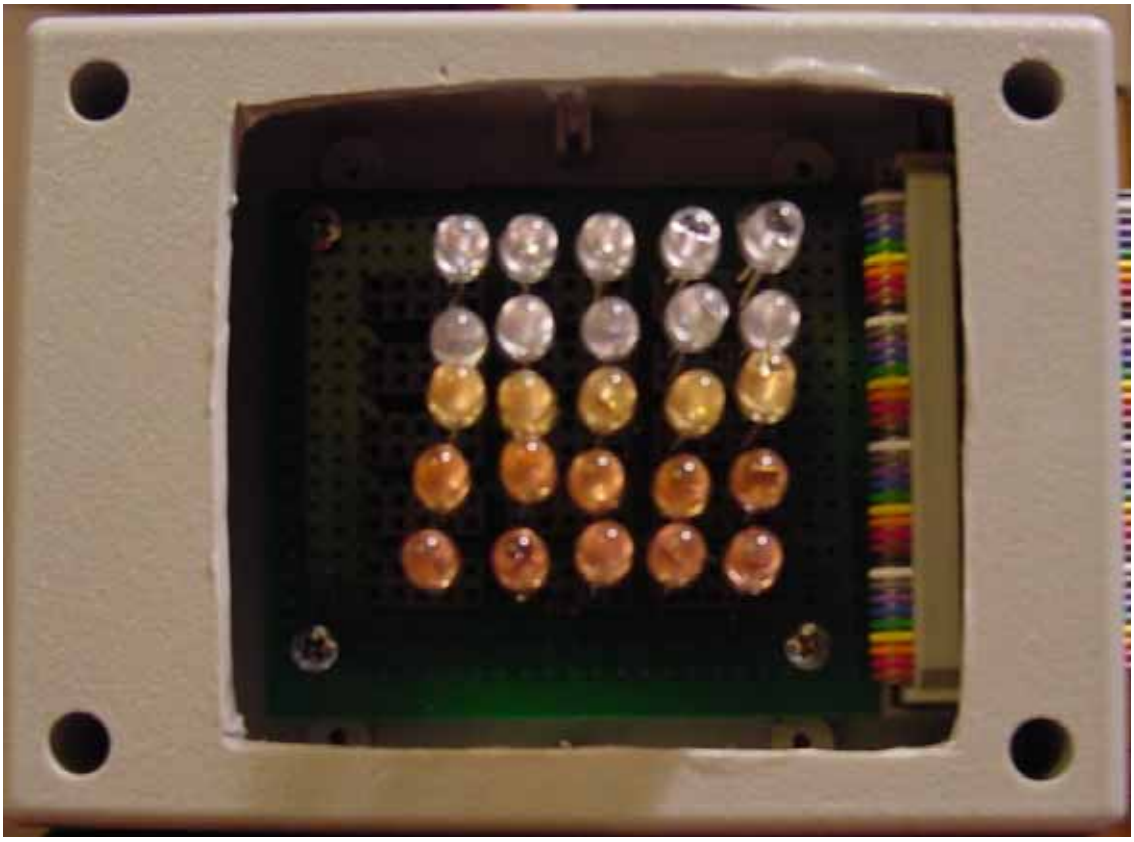

Figure 4.10: SSSI-II diode head without collimating cover in place.

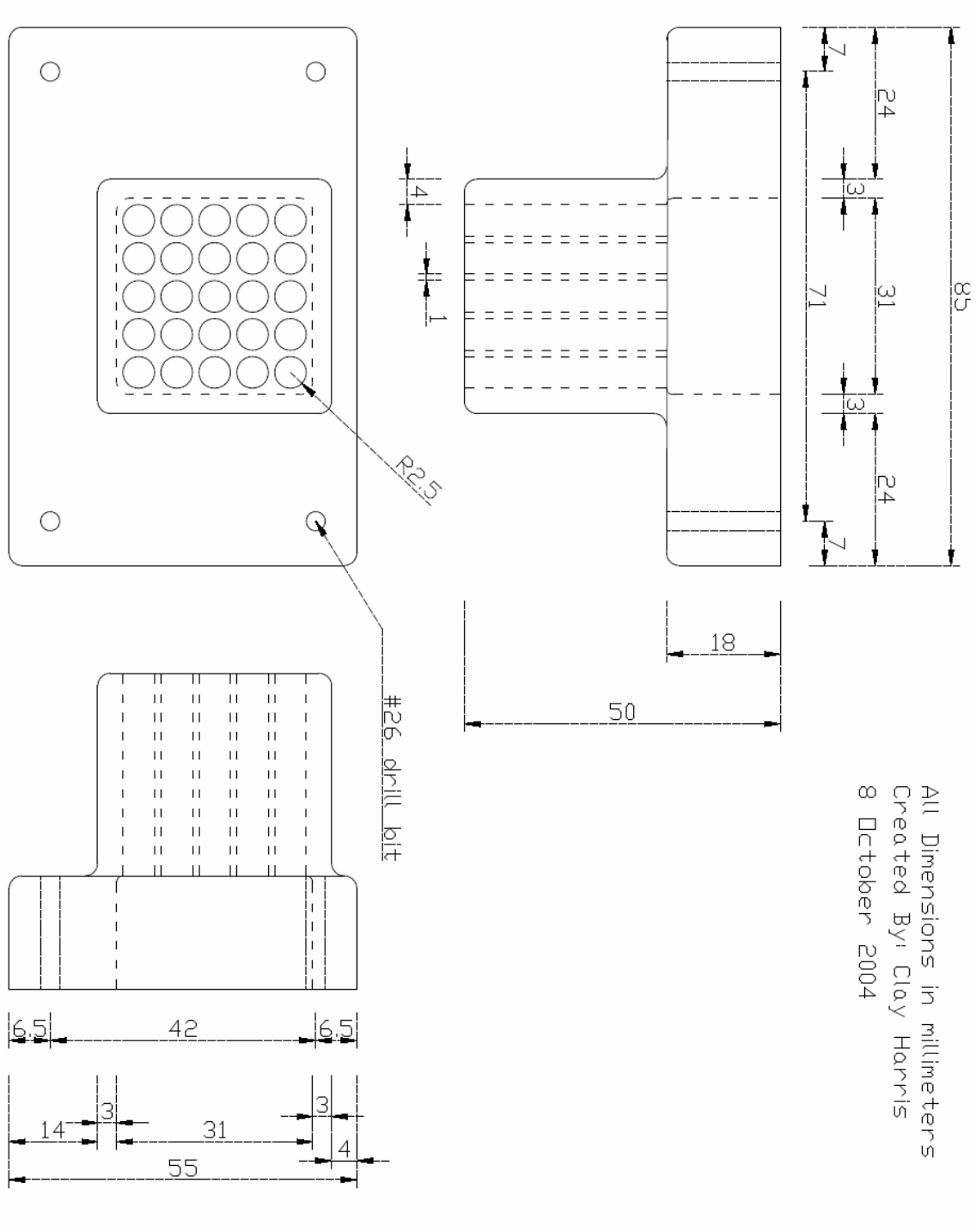

Figure 4.11: SSSI-II collimating cover.

Figure 4.12: SSSI-II diode head with collimating cover in place.

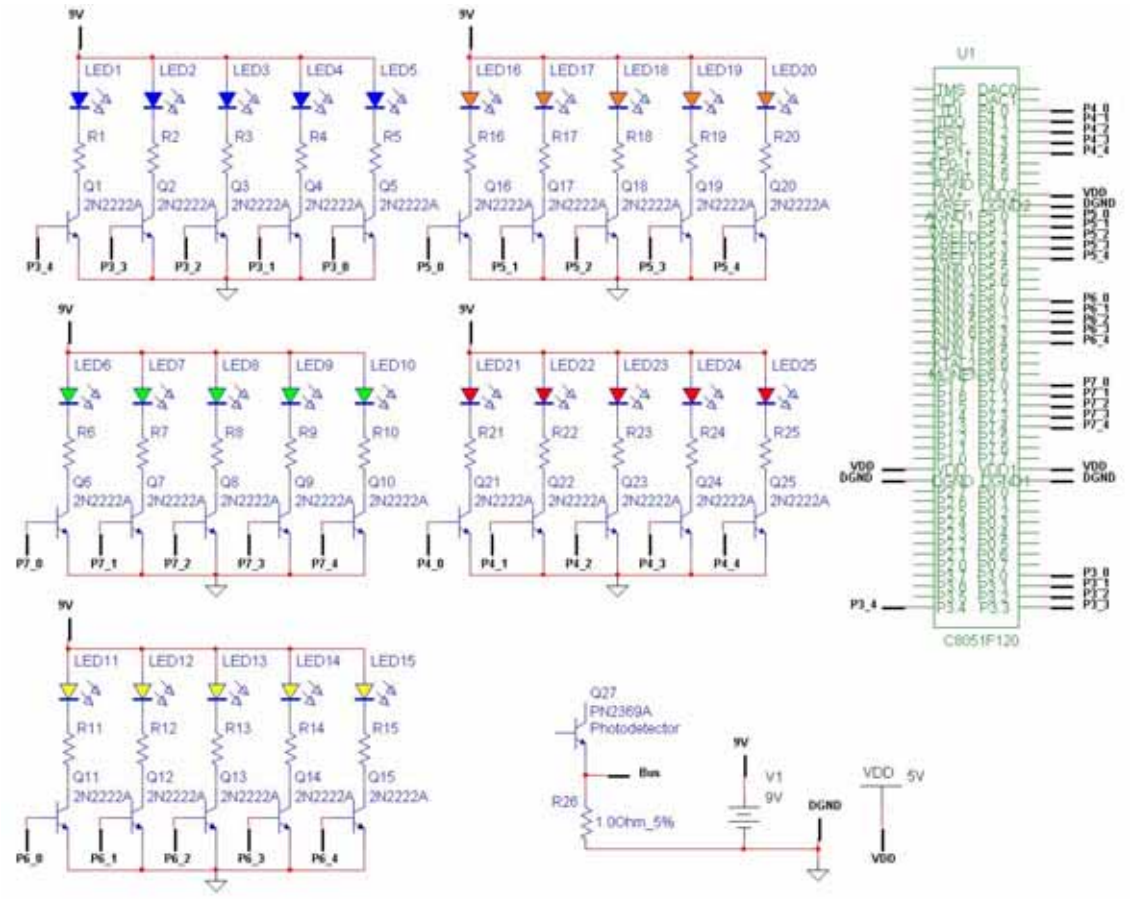

Figure 4.13: Circuit diagrams for the circuitry from the controller to the diode head as well as the detection circuitry.

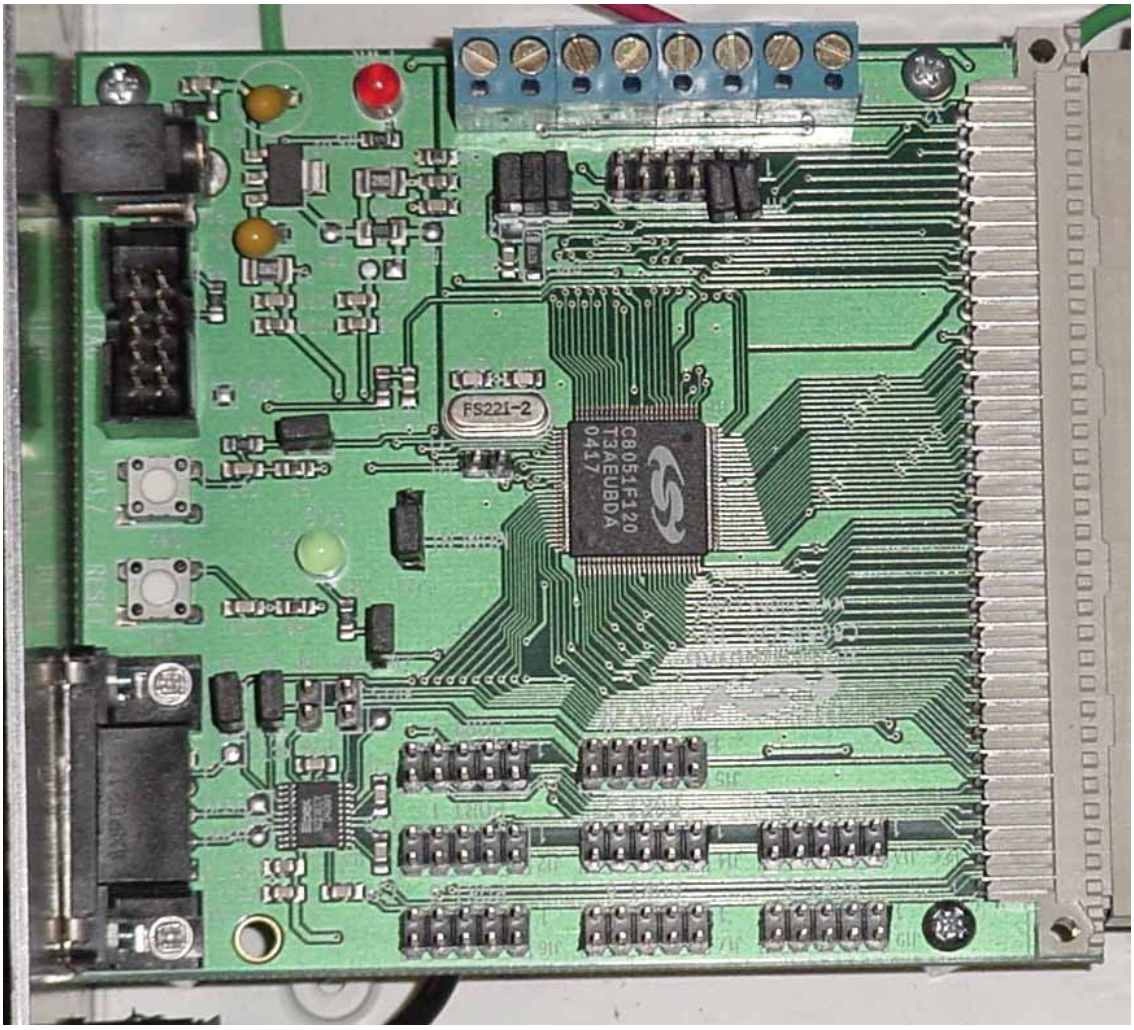

Figure 4.14: SSSI-II controller board.

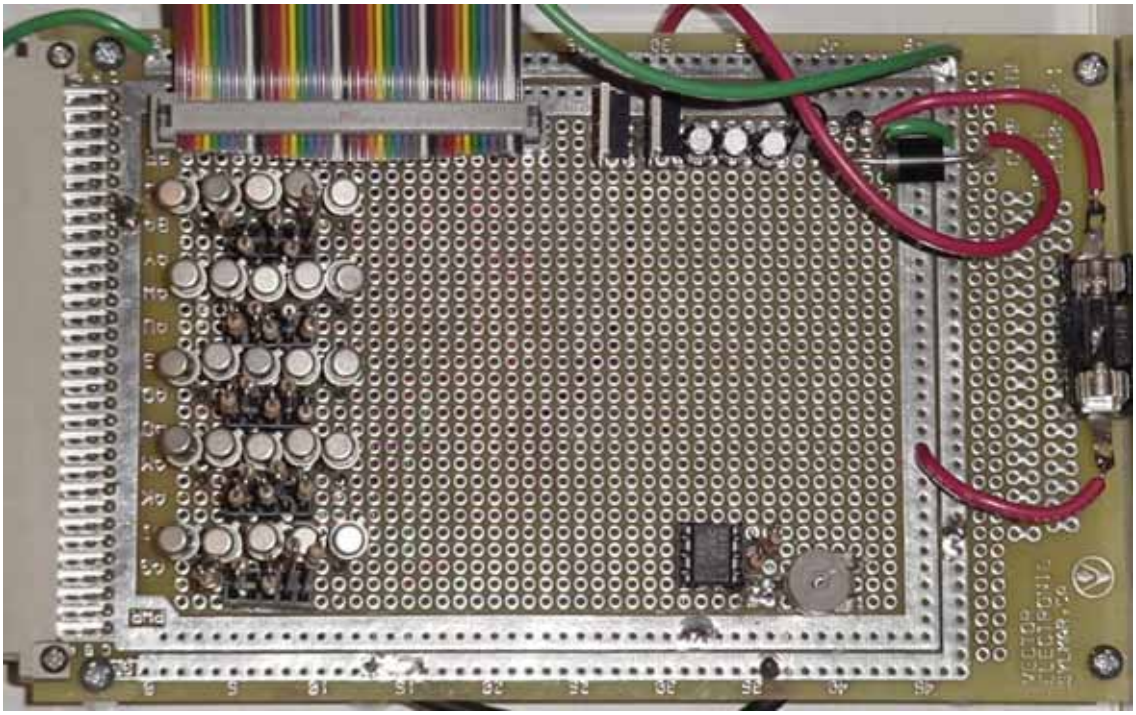

Figure 4.15: SSSI-II expansion board with circuitry for switching on and off the LEDs, detection circuitry, and voltage regulator circuitry.

Figure 4.16: Connection end of the SSSI-II housing.

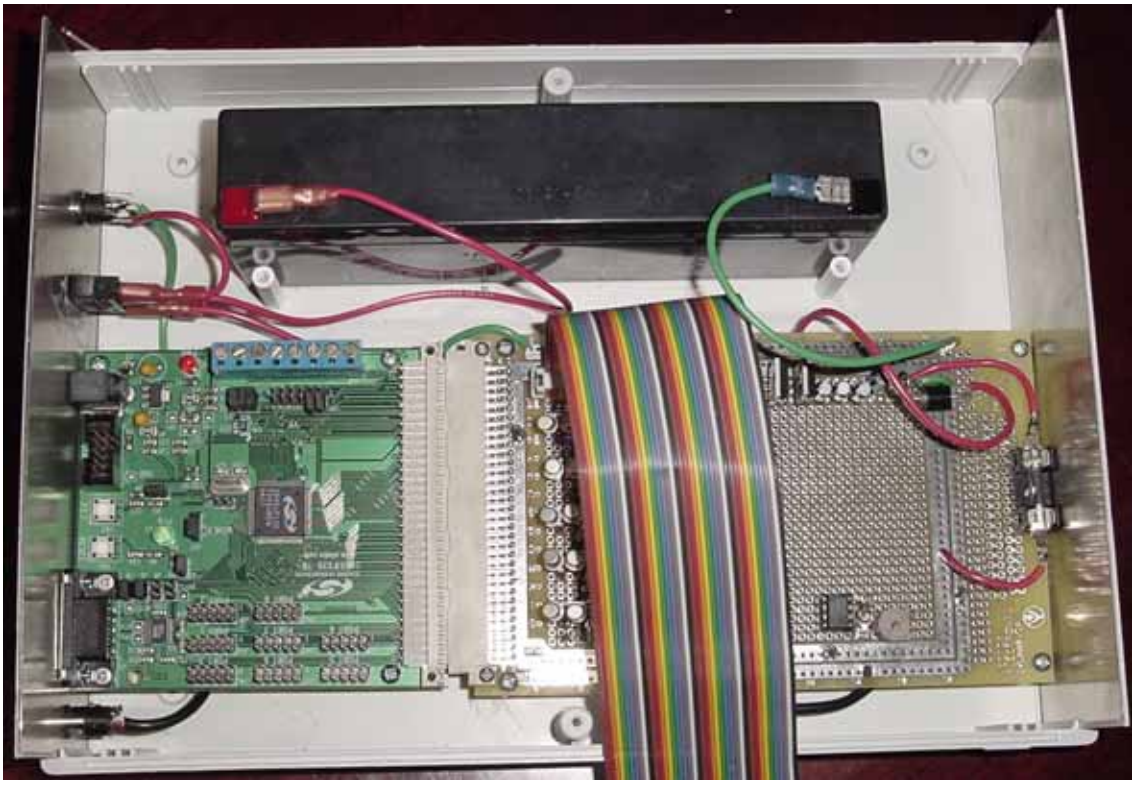

Figure 4.17: Inside the SSSI-II.

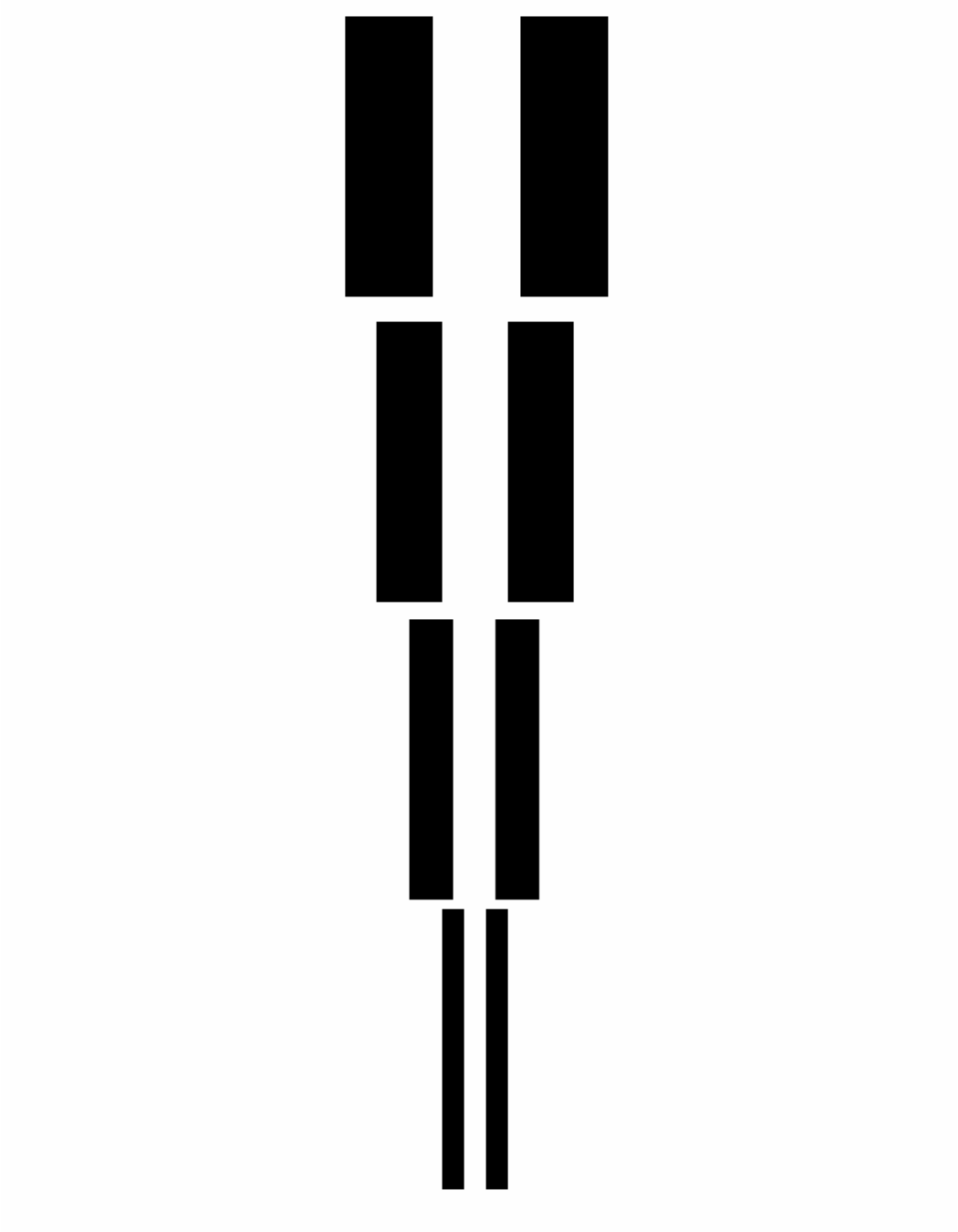

Figure 4.18: Spatial resolution test pattern.

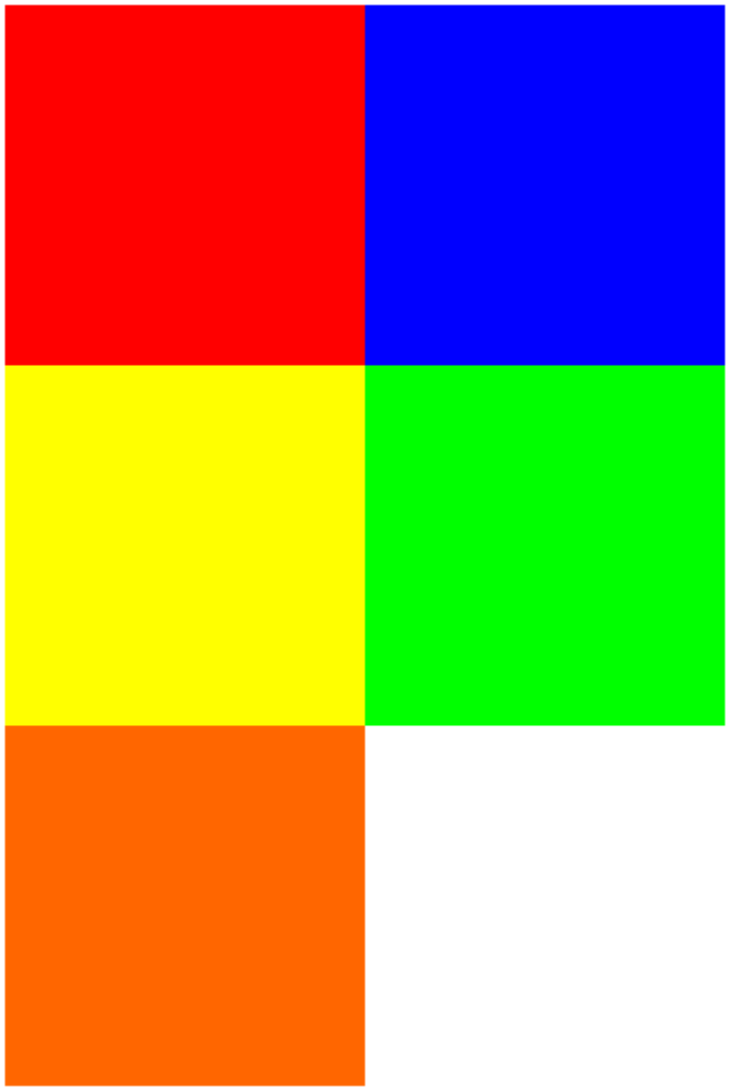

Figure 4.19: Spectral resolution test pattern.

Figure 4.20: City Creek Canyon Nature Preserve water sampling with SSSI-II.

Figure 4.21: Great Salt Lake Marina water sampling with SSSI-II.

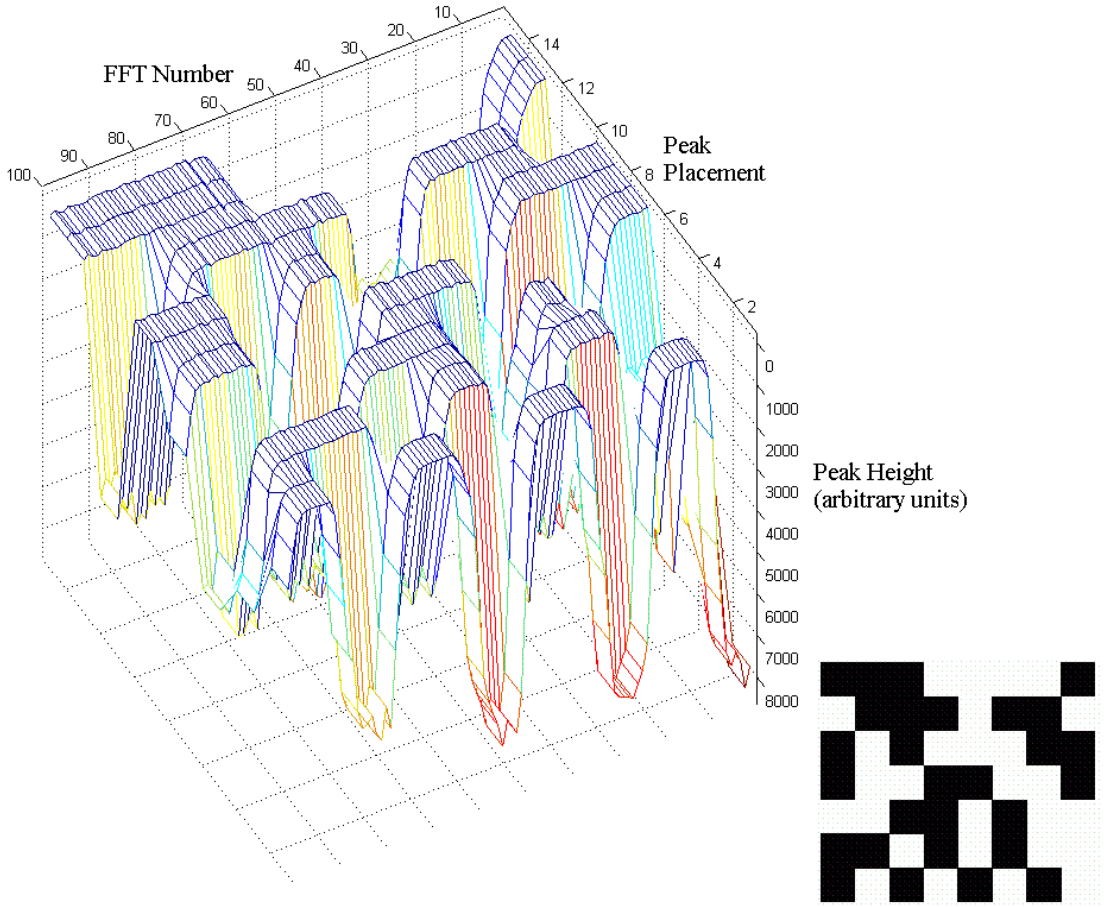

Figure 4.22: Test Grid Results (left) with template (right) for ease of comparison.

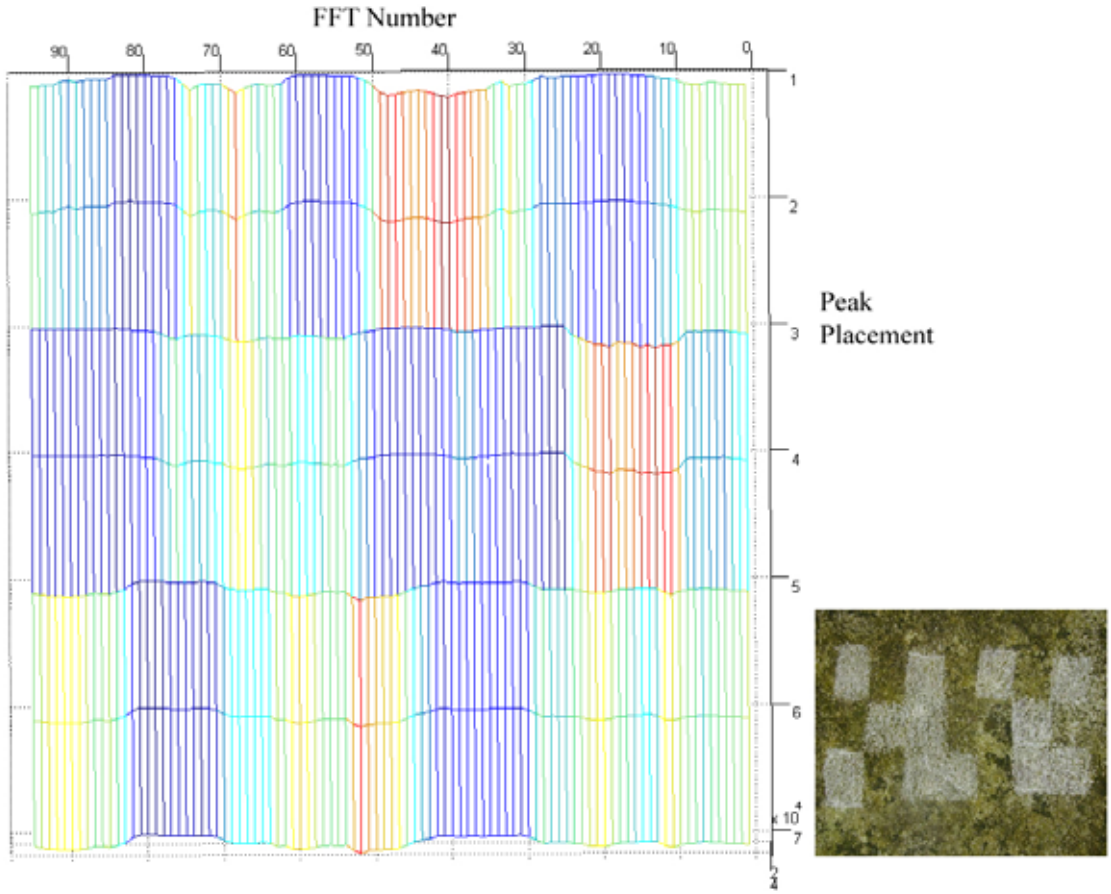

Figure 4.23: Grid etched into black and green biofilm on concrete (right) scanned with the SSSI-I gave the results (left).

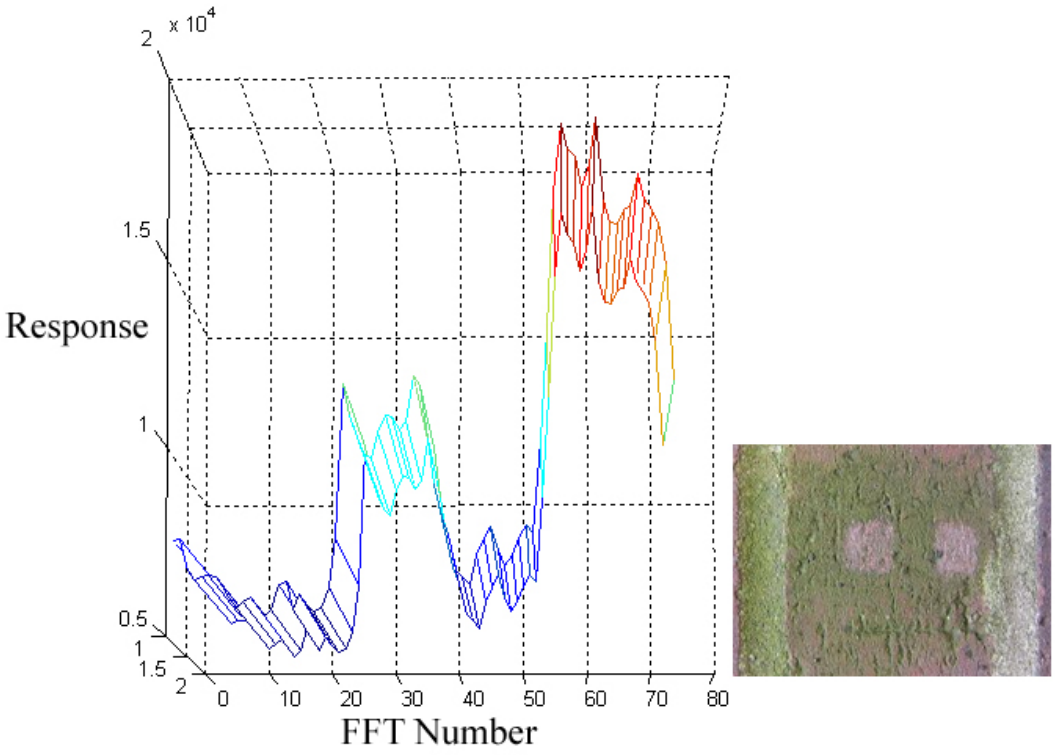

Figure 4.24: Grid etched into green biofilm on red brick (right) scanned with the SSSI-I gave the results (left).

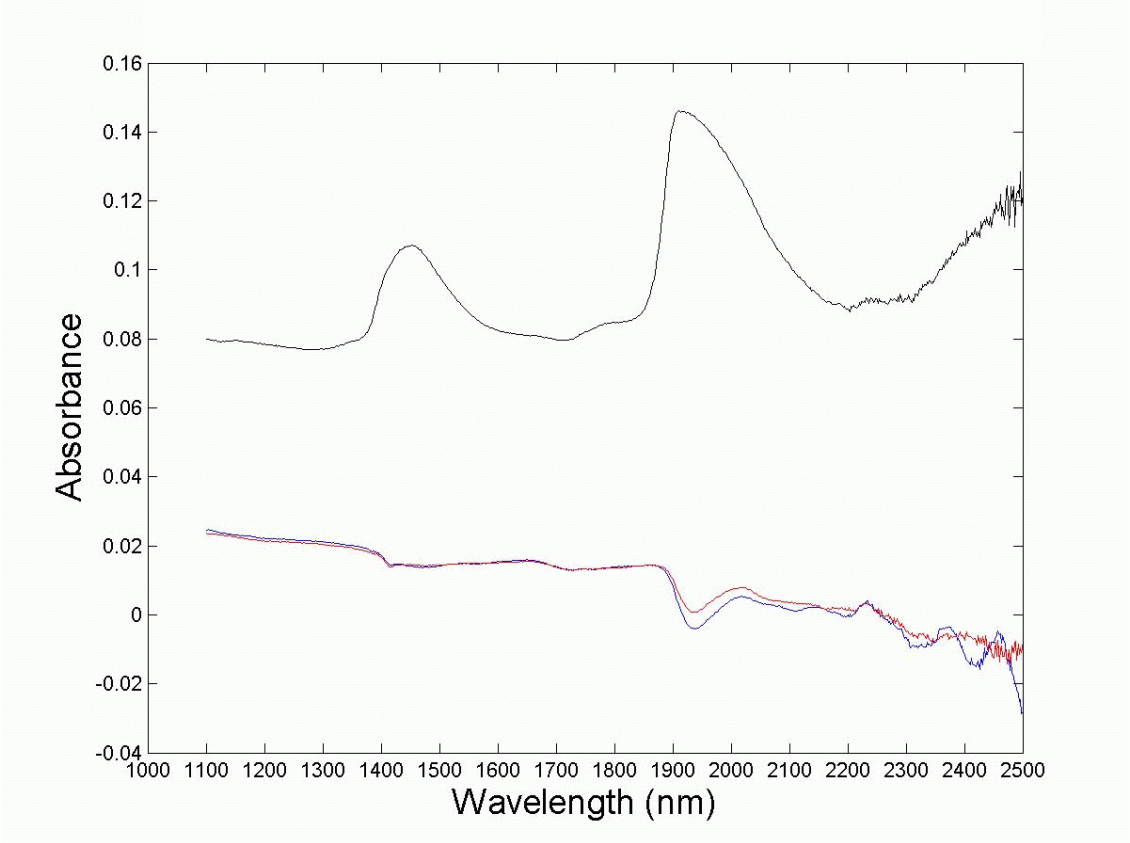

Figure 4.25: Average background corrected spectrum of Gloeocapsa biofilm on limestone before (black) and after baking one (blue) and two (red) times at 160° C for three hours.

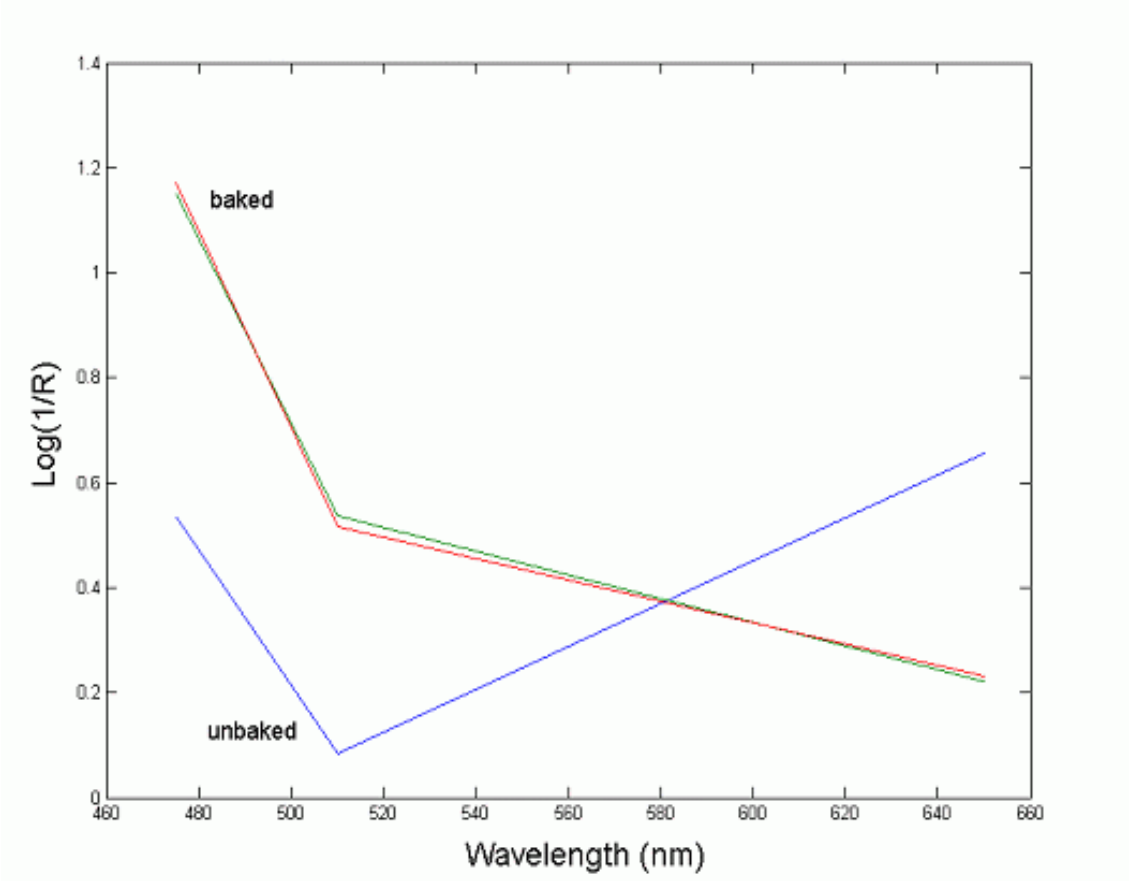

Figure 4.26: Average background corrected visible spectrum with three diodes of Gloeocapsa biofilm on limestone before (black) and after baking one (blue) and two (red) times at 160° C for three hours.

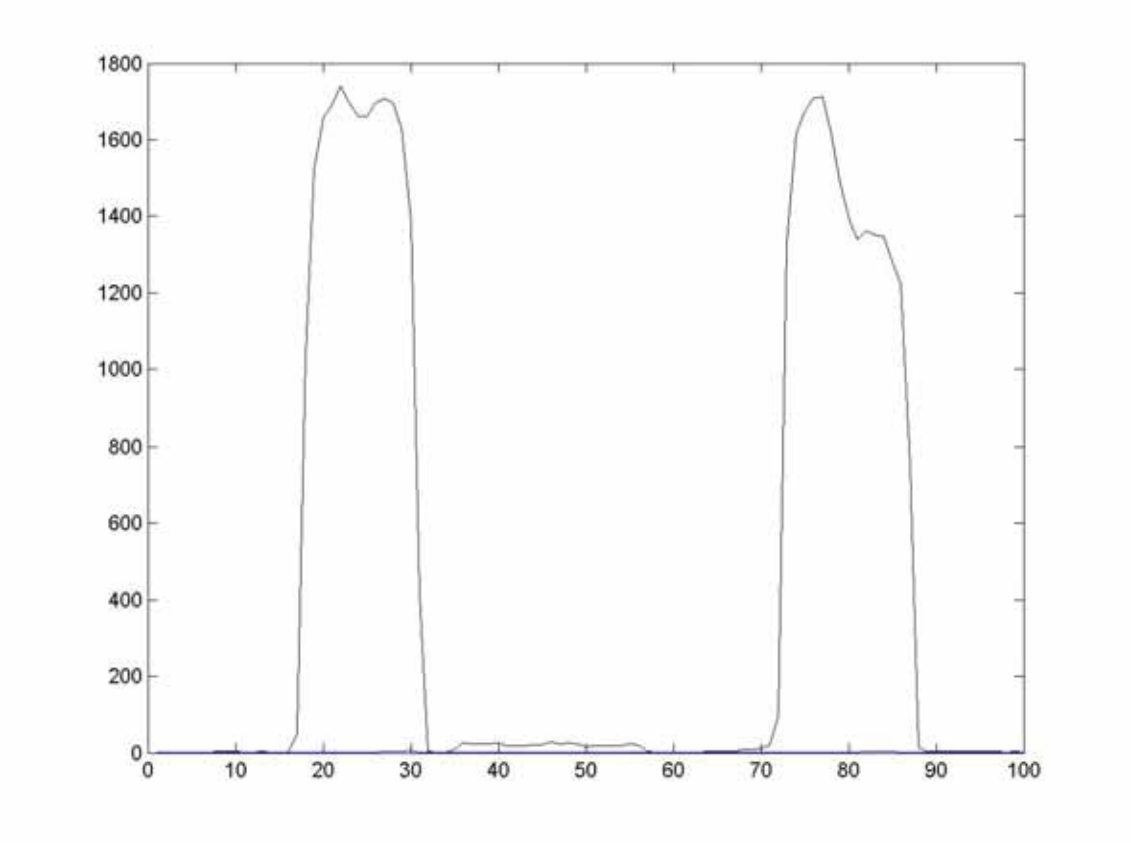

Figure 4.27: Response of a COTS laser diode vs. FFT number plot of the black-white-black-white-black target in Figure **4.8** from 10 m.

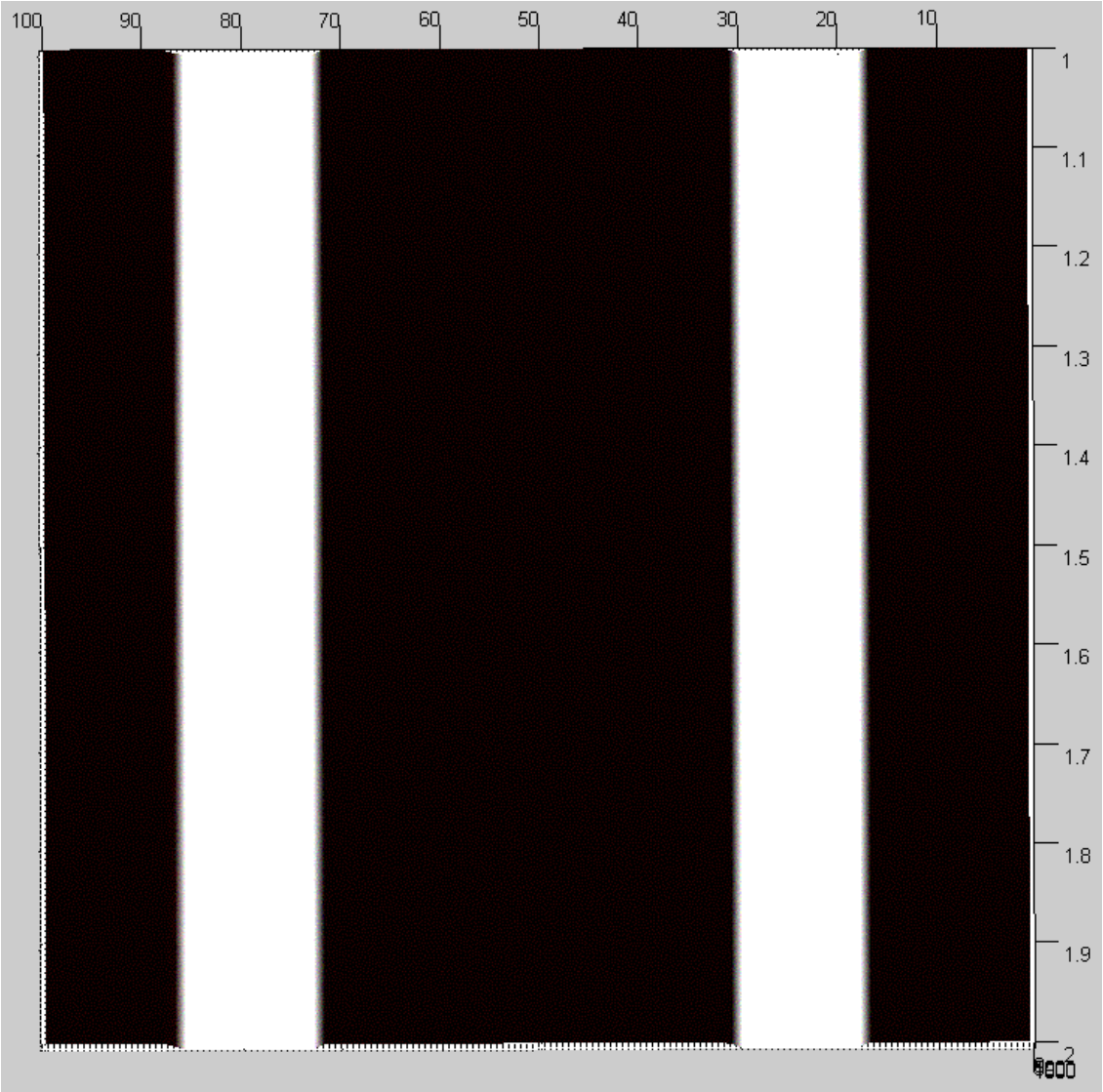

Figure 4.28: Reconstructed image in grayscale from the COTS laser diode response after scanning the black-white-black-white-black target in Figure **4.8** from 10 m.

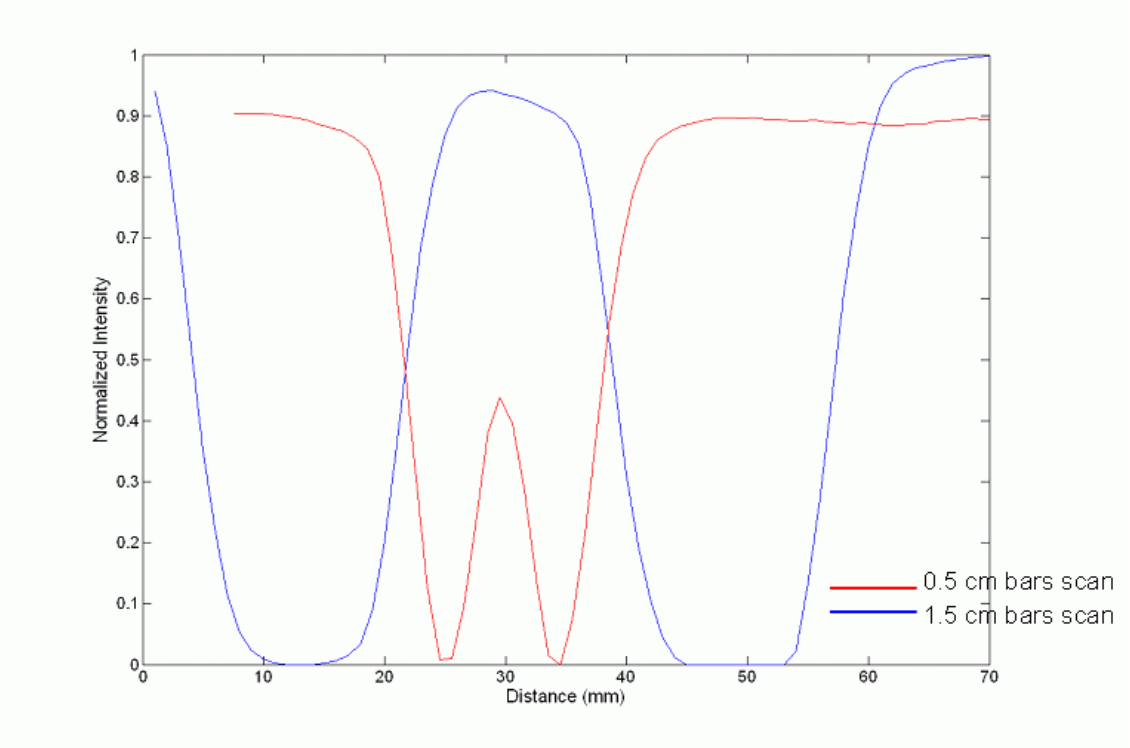

Figure 4.29: Spatial resolution test pattern results. 1.5 cm and 0.5 cm resolution tests indicate the ability to resolve 5 mm features.

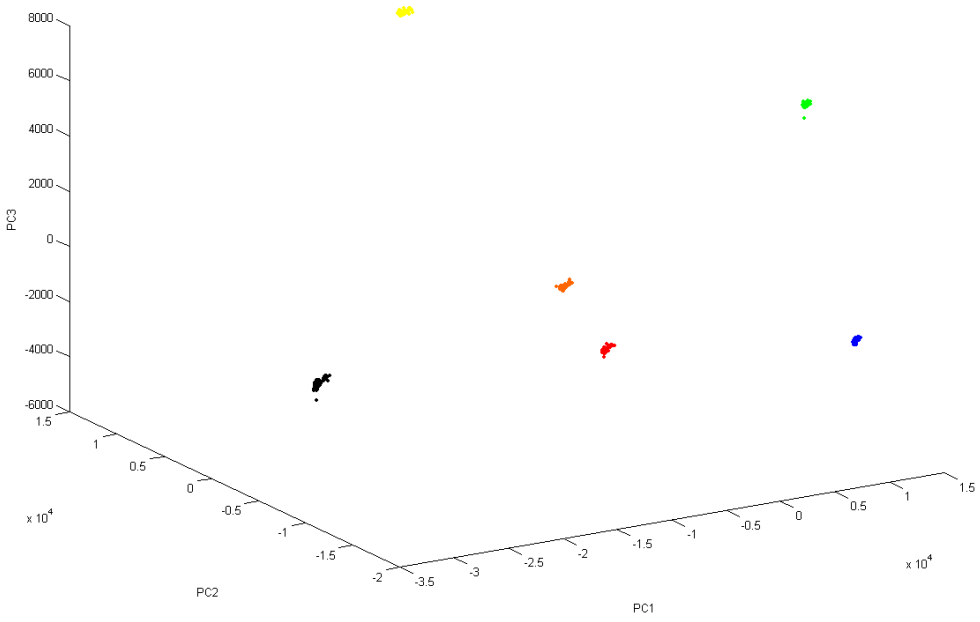

Figure 4.30: Principal Component Analysis of the six colored reflectance standards with data point color correlating with the color of the standard (black points used for white standard).

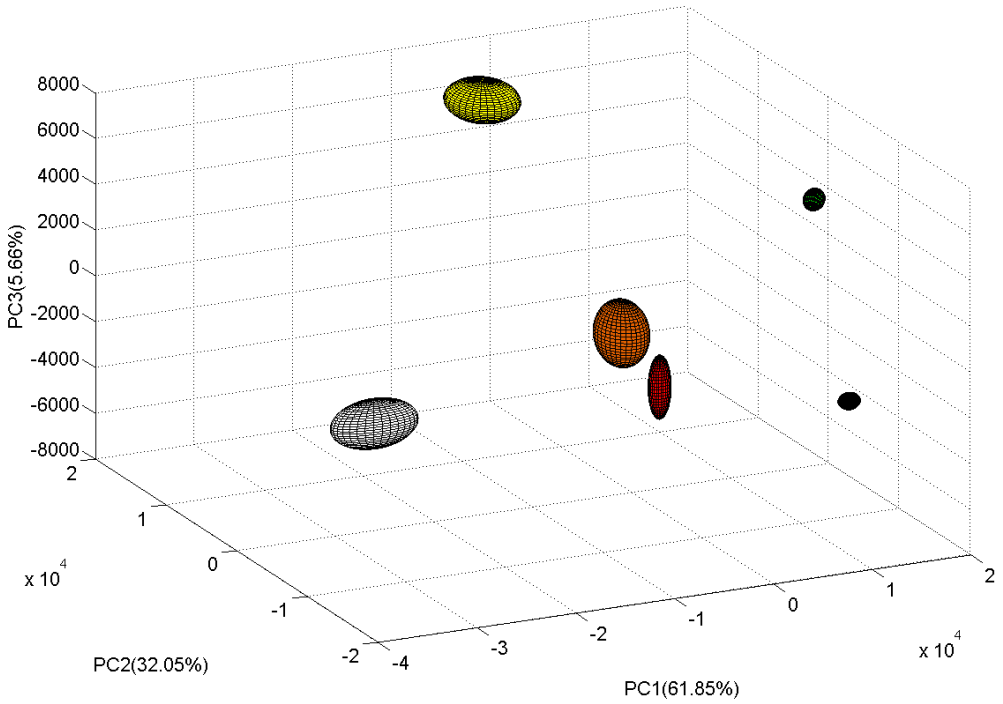

Figure 4.31: Principal Component Analysis of the six reflectance standards with 15 STD ellipsoids surrounding data clusters. Ellipsoid color correlates with the color of the standard.

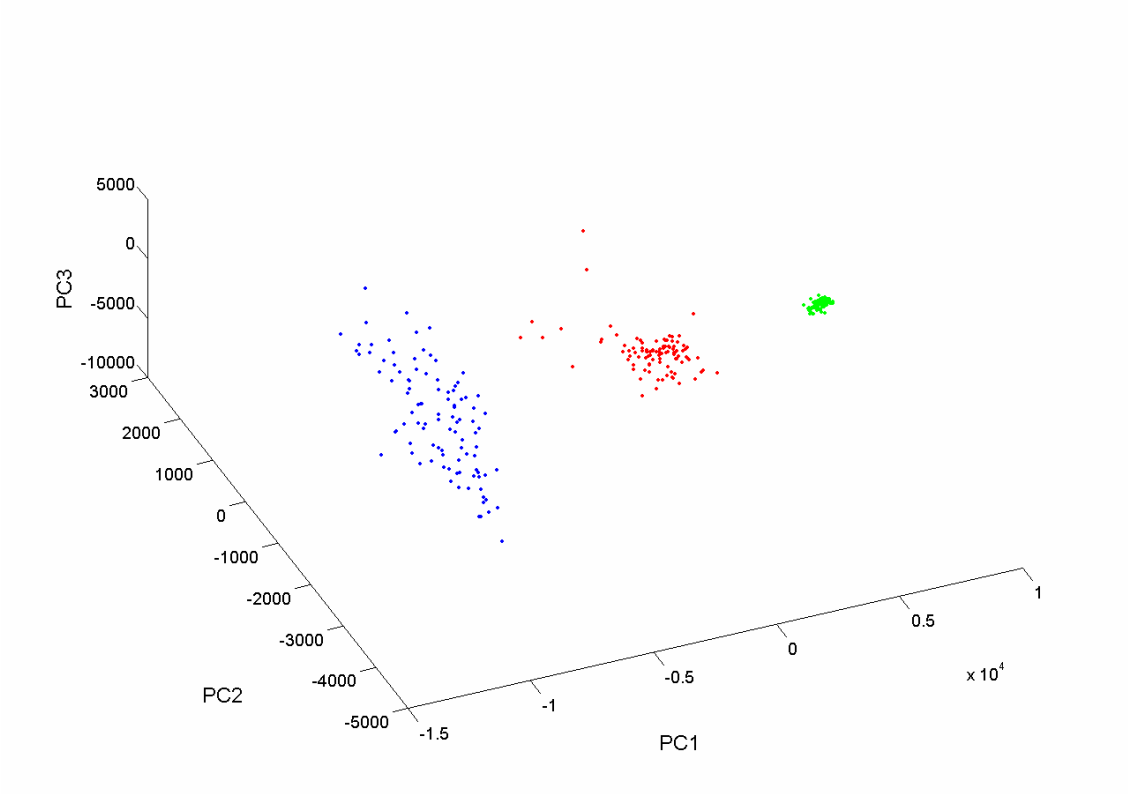

Figure 4.32: Principal Component Analysis results of data collected near Salt Lake City, Utah. Green is data collected in the City Creek Canyon Nature Preserve Exit Water, red is data collected from the Hensley/Salt Lake Cutoff – Warm Spring Park, and blue is data collected from the Great Salt Lake Marina.

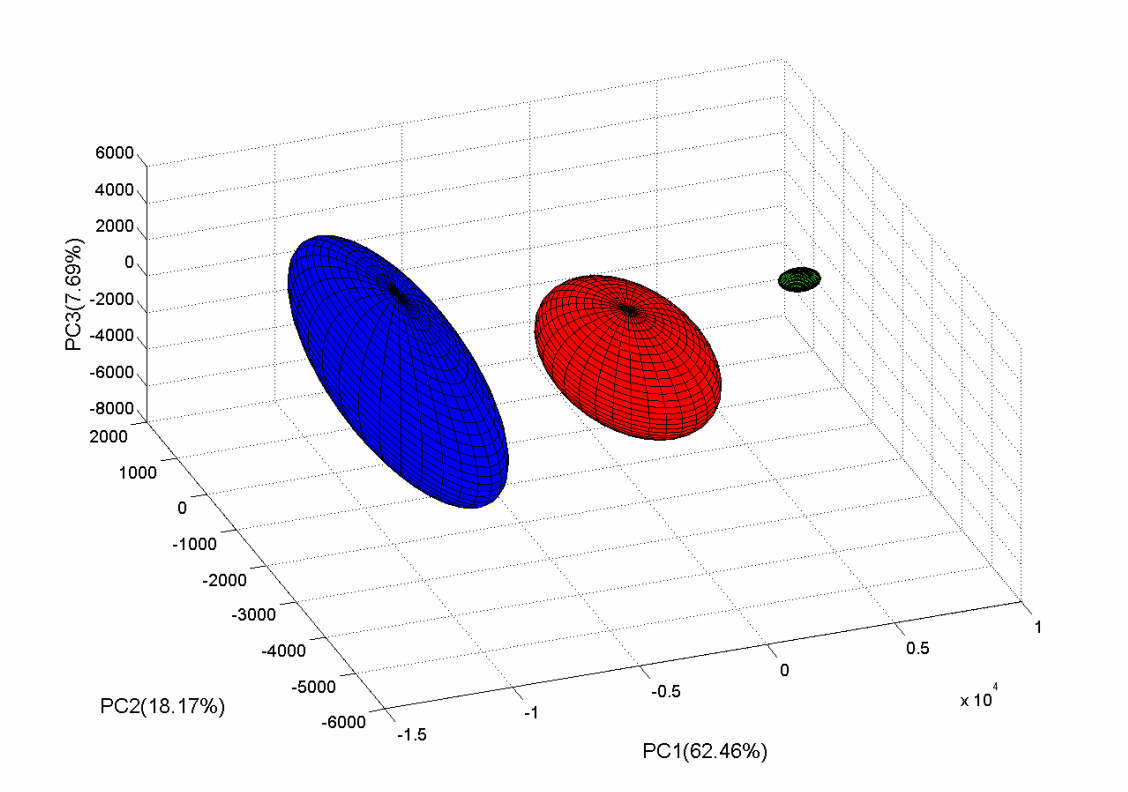

Figure 4.33: Principal Component Analysis results of data collected near Salt Lake City, Utah with 15 STD ellipsoids surrounding data clusters. Green is data collected in the City Creek Canyon Nature Preserve Exit Water, red is data collected from the Hensley/Salt Lake Cutoff – Warm Spring Park, and blue is data collected from the Great Salt Lake Marina.

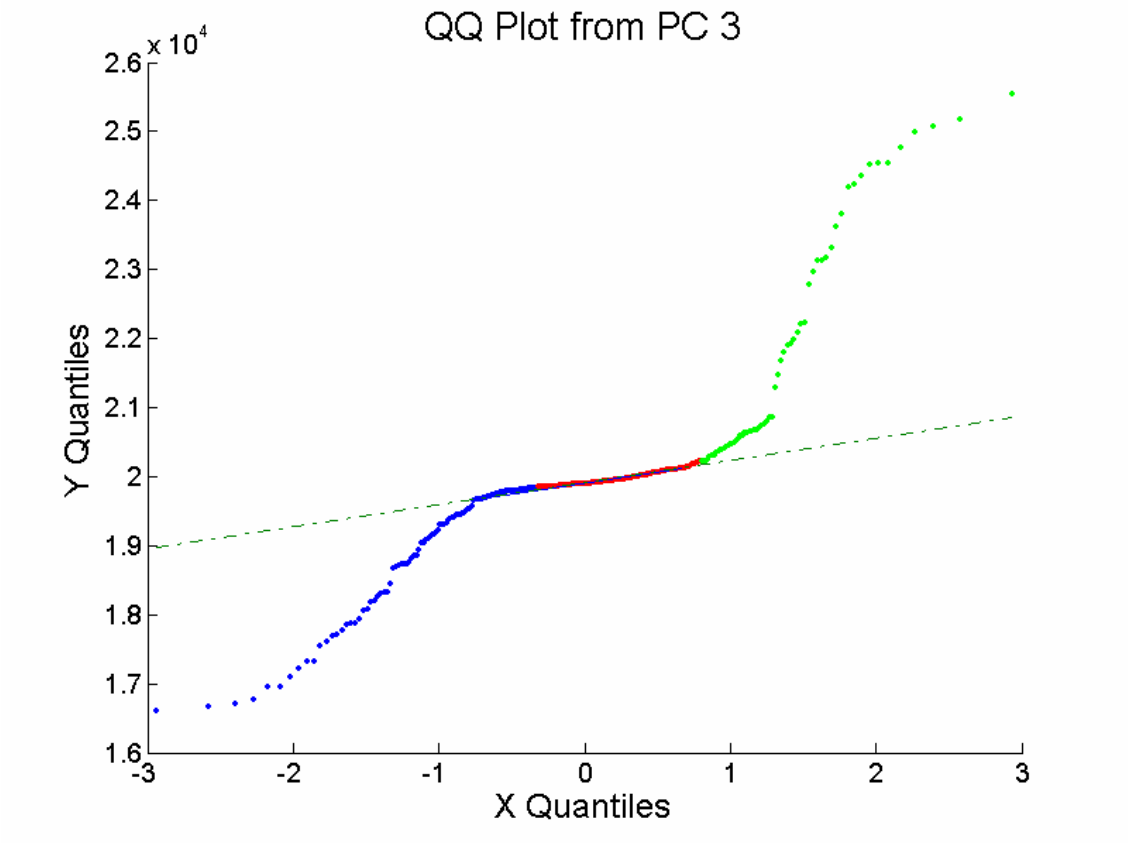

Figure 4.34: QQ plot produced by SAQQ analysis of PC3. Clusters defined by SAQQ are indicated by color. Green is data collected in the City Creek Canyon Nature Preserve Exit Water, red is data collected from the Hensley/Salt Lake Cutoff – Warm Spring Park, and blue is data collected from the Great Salt Lake Marina.

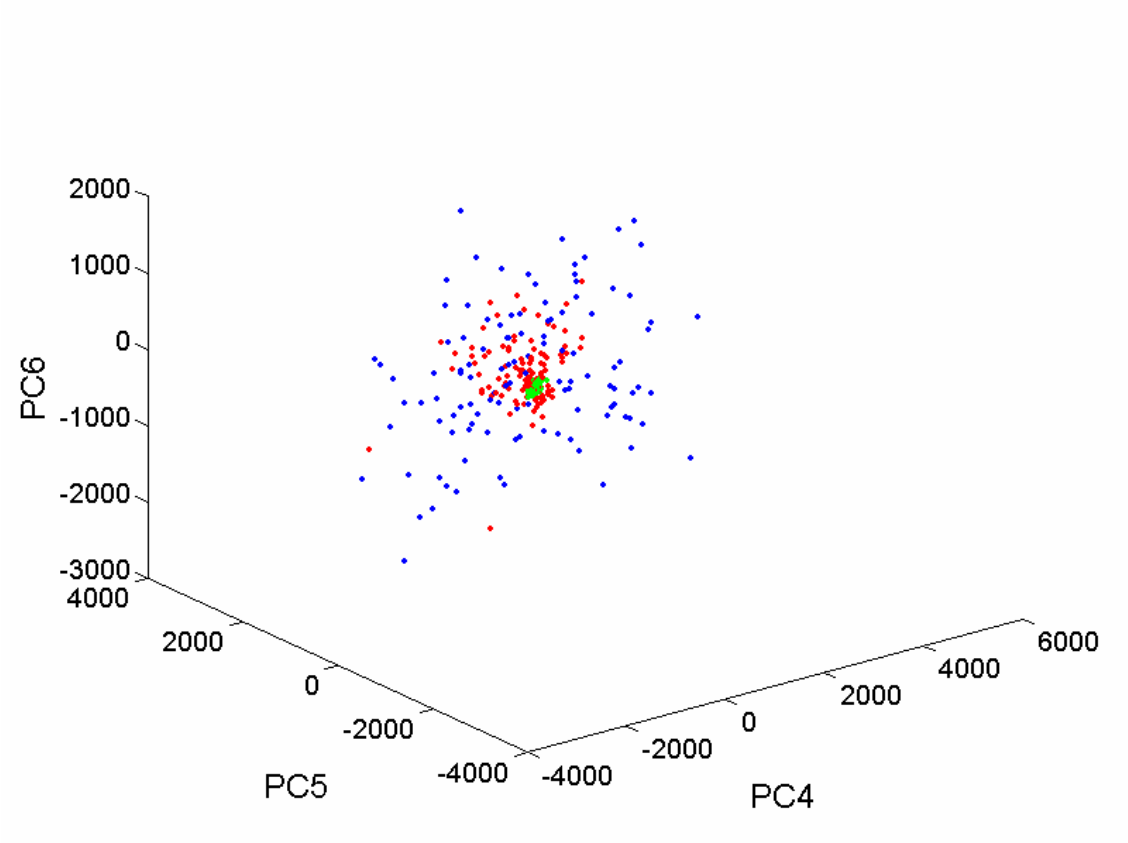

Figure 4.35: Clusters defined by SAQQ analysis of PC3. Clusters defined by SAQQ are indicated by color. Green is data collected in the City Creek Canyon Nature Preserve Exit Water, red is data collected from the Hensley/Salt Lake Cutoff – Warm Spring Park, and blue is data collected from the Great Salt Lake Marina.

Copyright Statement

Douglas, C. C.; Harris, J. C.; Iskandarani, M.; Johnson, C. R.; Lodder, R. A.; Parker, S. G.; Cole, M. J.; Ewing, R.; Efendiev, Y.; Lazarov, R.; Qin, G. In *Dynamic Contaminant Identification in Water*, ICCS, 2006; Alexandrov, V. N.; Albada, G. D. v.; Sloot, P. M. A.; Dongarra, J. J., Eds. Springer-Verlag: 2006; pp 393-400.

Chapter Five – Integrated Computational Imaging with a Near-Infrared Near-field Scanning Optical Microscope (ICI NIR-NSOM)

Introduction

Near-infrared (NIR) microspectrometry is an established technology for acquiring spectral data on micron sized samples. Traditional NIR spectrometry has benefits over IR spectrometry in that water is transparent to NIR allowing for aqueous samples such as *in vivo* catheter studies. However, conventional IR and NIR microscopes limit spatial resolution to about $\lambda/2$, making the best NIR spatial resolution 0.5-1 μm[90]. The spatial details in NIR microspectroscopy become lost after light travels a wavelength from a feature smaller than $\lambda/2$, but with smaller wavelengths higher resolutions can be gained. To overcome this limitation, research is being directed toward the development of near-field NIR microscopic devices.

Utilization of the near-field effect is achieved when a subwavelength-sized optical element is placed at a subwavelength distance between the element and the sample. Because the light only travels over a short distance, diffraction and far-field characteristics are eliminated. This can allow for much higher resolutions. Resolution of 80-120 nm for a light source of 1200 nm has been reported[90].

Near-field Scanning Optical Microscopes (NSOM) are currently being used as a method for improving spatial resolution below the wavelength of light. Since there is a high loss in signal when light passes through subwavelength apertures, NSOM is frequently

employed where there is a very high signal-to-noise ratio and a need for subwavelength spatial resolution. Most IR samples have quite low molar absorption values leading to low signal-to-noise ratios in IR NSOM. NIR's weak overtones and combination bands are even weaker than the IR fundamentals. Ultimately all of these factors add up in NIR-NSOM to require days of tedious scans for mapping any sizable sample area (10 x 10 μm). This in turn leads to the infrequent and difficult use of IR and NIR NSOM.

The first method for optimizing NSOM is by increasing the intensity of the light source. The synchrotron has an intense collimated beam similar to a laser that is ~1000 times the brightness of a conventional infrared source[85]. Wetzel *et al.* state that scanning with the synchrotron source is 30 times faster than with a globar source. The intensely bright synchrotron will cause significantly more light to reach the sample and therefore the detector. The result will be increased signal-to-noise ratios and decreasing scan times.

The hyperspectral images obtained from scanning two-dimensional surfaces tend to be data rich and information poor. Scanning across the spectrum has increased the data volume creating a bottleneck in hyperspectral imaging, increasing the time it takes to analyze and interpret data. Several methods are being developed that both decrease the total volume of data and decrease the time it takes to interpret the data. These methods are often called hyperspectral integrated computational imaging (HICI) or integrated sensing and processing (ISP).

Using ICI, computers both capture and analyze an image in the same process. ICI reduces the data load to the computer by downloading some of the computations to the image sensor itself[90]. On the other hand, HICI is the process by which a spectral image is encoded at many different wavelengths simultaneously. The resulting intensity signal at the detector is then directly proportional to the analyte(s) encoded in HICI. Both ICI and HICI incorporate Felgett's (multiplex) advantage through collecting all wavelengths at once. This in turn reduces the total time required for obtaining the same signal-to-noise ratio when compared to a dispersive technique. HICI methods for collecting data are advantageous because they compress a high volume of information into a single image scan, which saves time and decreases the final volume of data collected.

Through the ISP process, the steps by which signals are detected and interpreted are being completely reorganized, reworked, and optimized. This project seeks to treat the total structure as a single end-to-end process that can be optimized globally. ISP has created new mathematical algorithms and produced new statistical methods by which both the speed of sensing and amount of data are increasing, and the information gained is more meaningful. By developing and implementing new ICI and ISP methods and instruments, the bandwidth, volume of data to be interpreted, and amount of time taken to interpret data are each being significantly reduced, while maintaining a high level of meaningful information[90].

Principal components (PCs) of spectral data are formed from loadings vectors. The highest loadings correspond to wavelengths where the variation of interest in the sample

spectra is greatest. The variations can be captured optically by selecting molecular filter (MF) substances with transmission spectra similar to the loadings. If there are positive and negative loadings in the MF bandpass, two MFs must be employed for that PC to avoid ambiguity. The process of using MFs to compute and filter the incident light for the analyte of interest is called Molecular Factor Computing (MFC). MFC reduces the total quantity of data collected and therefore the time it takes to collect the data, by targeting only the analyte(s) of interest.

Abdominal aortic aneurysms (AAAs) occur in 2-8% of people over age 60 in the United States. Recently developed knockout mouse models -have made it possible to study the occurrence of aneurysms more closely[91]. Infusion of AngII results in the creation of aortic aneurysms as early as 7 days after the onset of AngIl delivery, with the majority of results obtained in mice infused with AngII for 28 days[92]. A number of studies have shown a trend of decreasing elastin and increasing collagen leading to the development of an AAA[92]. A similar trend has been reported in the thin fibrous cap of vulnerable atherosclerotic plaques[85]. In this region, plaque rupture appears to be initiated in a region smaller than 5 x 5 μm, posing a problem for study by conventional microspectrometry.

Analysis of AngII infused mice has raised many questions about the current animal model. Of particular interest is the consistent localization of AAA formation above the kidney. Utilizing MFC NIR-NSOM, this study analyzes the elastin, collagen I, and collagen III content above and below the kidney in both a control aorta and in an aorta which has undergone aneurysm.

Experimental

Tissue Samples.

This study used apolipoprotein E (apoE) -/- knockout mice ranging in age from 2 - 12 months. Infusion of angiotensin II or saline by osmotic mini-pump into the subcutaneous space of mice took place at doses ranging from 500 to 1,000 ng/kg/min for 7 - 28 days. 2 mm sections from above (ak) and from below (bk) the kidney branch of the aorta from both an experimental mouse (x) and a control mouse (c) were frozen in OCT. The 2 mm sections were sequentially sectioned from top to bottom into 12 consecutive 5 μm slices resulting in 48 total samples (x-ak-1 to 12, x-bk-1 to 12, c-ak-1 to 12, and c-bk-1 to 12). These samples were mounted on two low-e glass slides (SensIR) for 2-D FTIR reflectance spectrometry. An aneurysm that was large and grossly discernible occurred above the kidney branch in the experimental sample, while only vessel wall thickening was observed below the kidney branch. All sections above and below the kidney branch in the control sample were considered normal. A visible light image of the first section of all four groups appears in Figure 3.1. All procedures involving animals were approved by the Animal Care and Use Committee at the University of Kentucky.

Instrument Construction.

In this study, the near-field effect will be employed by utilizing a quartz fiber optic probe as the emitting aperture. The aperture, which begins with a 5 μm core, is pulled to smaller than half the largest wavelength, and placed less than half a wavelength from the surface of the sample. This is achieved by threading a fiberoptic fiber through a thin brass tube. A one gram weight is attached to the bottom end as a pulling weight while

the top end is held. A propane torch is used to heat the tubing and thus the fiber on the inside. Ultimately, the fiber pulls into two pieces because of the added weight and heat. Guides prevent the fiber from coming in contact with the heated tube as they are quickly removed. To prevent light from escaping the fiber optic cladding where it has been pulled, it is coated with silver by using Tollen's reagent.

The NSOM box was custom designed and was approved as safe by the safety officers at the National Synchrotron Light Source located at Brookhaven National Laboratory in Upton, NY. Figures 5.1-5.8 show the different portions of the NSOM box and are dimensioned in the figures. The overall box size is approximately 1 m x 0.75 m x 0.5 m. The NSOM (See Appendix C for complete NSOM Manual) enclosure consists of two separate chambers (Figures 5.8-5.9). The synchrotron is coupled to the first chamber, allowing synchrotron light to enter into the enclosure. In this light coupling side of the NSOM box (the larger half), there is a 12" focusing lens (Thor labs lens LB1917 dia = 50.8 mm efl = 300.0 mm), 3-D micrometer resolution stage (Melles Griot 17AMB03/R), a standard fiberoptic chuck (Back-Loading Collet Style Fiber Chuck (Thor Labs HFC005)), and an Adjustable Flexure Fiber Chuck Holder (Thor Labs HFG003). The layout is shown in Figures 5.10-5.11. The fiberoptic is fed through the fiber chuck and then inserted into the fiber chuck mount (frequently easier to simply tape the fiber to the chuck). The incoming beam is coupled at the focus point of the light by moving the 3D stage, fiber chuck, and the chuck mount. Correct alignment including pitch and yaw is required for the best signal. Molecular filters are placed in the beam path before the synchrotron light passes into the quartz fiber. On the detection side of the NSOM box,

there is a 3-D stage (Newport M-461 series) mounted on an optical rod for holding the pulled end of the fiberoptic, a Melles Griot Nanometer Stage (NanoBlock-M SN-500026), a Large Area (1 mm) InGaAs Photodiode detector (Fermionics FD1000W), a WinRadio (WinRadio Communications WR-1500e), and a SoundBlaster (model #SB0270) sound card (Figures 5.12-5.13). The fiber is attached to the 3-D stage by taping it with cellulose tape (Scotch tape) to a rubber stopper (to help dampen vibrations), which is in turn mounted on the stage. The height and position of the fiber tip may be adjusted with the 3-D stage and there are also coarse adjustments for all three axis on the Nanometer Stage. Gentle tapping of either stage induces vibrations in the tip of the fiber probe. Dampening of these vibrations may be visibly noticed under 80X microscope magnification and indicate close proximity of the tip to the sample. The distance between the pulled tip of the quartz fiber is brought to within 500 nm of the sample using fine adjustments on the nanometer stage. Directly behind the sample is an InGaAs photodiode detector. This detector was chosen for its rapid rise and fall time, which enabled locking in to the synchrotron's 52.887 MHz pulse rate. The photodiode is connected to the WinRadio via a 1' BNC cable connector. The WinRadio is used to lock into the synchrotron's pulse and is tuned to 52.987 MHz on lower sideband mode (LSB) to yield a 1 kHz tone. The WinRadio is controlled by the computer via a standard RS232 9-pin serial cable or with a USB to serial adapter. The WinRadio may be disconnected from the computer's serial port after it is configured. Power for the WinRadio is provided by a 12 VDC source (EverStart battery 350 CA – 275 CCA). A 1/8" stereo audio patch cable (DigiKey AE1268-ND) connects the WinRadio audio output to a Sound Blaster external sound card. Although the sound card may accept the 1/8" stereo connector, it has been

found that the RCA inputs on the sound card are less noisy. So, a 1/8" female stereo to two RCA male adapter may be employed if preferred. The sound card is connected to and powered by a USB cable connected to the computer's USB port. The SoundBlaster sound card is used as the analog to digital converter and all data is recorded directly in MatLab 7.0.1.

The nanometer stage movement is controlled with a Piezo Controller (ThorLabs MDT693A). The Piezo Controller is current limited and provides potentials up to 150 V. The potential across the piezos in this Melles Griot nanometer stage must not exceed 75 V. This potential may be limited both by a switch on the back of the piezo controller and in the software. The piezo controller is able to precisely control voltages to 0.1 V. This is equivalent to a 20 nm step. It is important to note that thermal movement within the Melles Griot nanometer stage is 1000 nm/° C so temperature control is mandatory. The Piezo Controller requires a serial connection to a computer. This may be connected via standard RS232 9-pin serial cable, or with a USB to serial adapter. Communication with the controller is established directly from MatLab through its COM port. The nanometer stage is connected to the piezo controller via three 75 V SMC transfer lines. BNC to PL259 adapters (Amateur Electronic Supply AIM BNC Female to PL259 – AIM 278120) are required at the back of the piezo controller. The SMC transfer lines are extremely sensitive to external noise. These cables and connections must be shielded and grounded carefully. Power is provided to the nanostage via the piezo controller, which is powered from a 120 VAC wall outlet. Currently, rough adjustments are completed by hand and fine adjustments are completed via piezo movement controlled in MatLab. Since data

acquisition with the NSOM is completely automated via MatLab, scan size in each direction, step size in each direction, and collection time per pixel may all be set independently.

The molecular filters placed in the far field of the detector are used to encode specific wavelengths of light prior to light passing into the quartz fiber. Materials are selected for use as molecular filters by comparing their transmission spectra to the factor loadings correlating to the analyte of interest. This produces voltages on detector elements that are directly proportional to analyte concentrations, and in turn images whose colors directly correlate to the concentration of the analyte.

Instrument Calibration.

The resolution of an NSOM instrument is dependent on the aperture size, source-sample separation distance, and scanning step size[93]. Utilizing the aperture as a source, this instrument design acts as a transmission-mode microscope, which raster scans the sample while a detector behind the sample collects the transmitted light. The instrument collects data cubes in which the x-y axis is the spatial coordinates of the sample, and the z axis is the spectral information. When a tunable laser source is used, the z axis is hundreds of elements deep. In contrast, in MFC-NSOM the z axis is typically 1-5 elements deep.

Building a microscope is only a portion of the problem. The results obtained with the instrument on reference samples must be tested against theoretical predictions. The initial predictions and measurements will be made on a wire SEM calibration grid. The

SEM calibration grid consists of parallel 50 nm in diameter wires that are spaced 497 nm apart on center. A visible image of a calibration grid (grey coating) and supporting structure (darker mesh) may be seen without a fiber (Figure 5.14), with an unaltered fiber tip (Figure 5.15) and with a pulled fiber tip (Figure 5.16). Data collection was completed with a step size of 20 nm in both the x and y directions. The data acquisition rate was set to 1 second per pixel. Scanning in the x direction incorporated 4000 nm or 200 pixels, while scanning in the y direction incorporated 1500 nm or 750 pixels. This gave a total of 15,000 data points. No filters were incorporated in this scan to achieve the most throughput and therefore the highest signal-to-noise ratio. Due to the properties of the quartz fiber and the sensitivity of the InGaAs detector, it is assumed that effects of light outside the NIR range are negligible. To make the model easy to manage, the following simplifications are employed:

1. The aperture under consideration is a subwavelength hole within a large metallic plate. In order to obtain an image, the aperture is scanned point-by-point in a plane close behind the sample.

2. The sample chosen is a metallic thin-wire. Thin-wire samples permit easy characterization of NSOM instruments. Instrument resolution can be determined based on the wire length and diameter. Polarization effects, unexpectedly shaped NSOM images and certain enhancement effects can also be easily explained using a thin-wire sample.

Moment Method Model.

The Moment Method (MM) model was employed as the computational model for result comparison. The MM model uses a constant magnetic surface current density within the

aperture, whereas the FDTD model uses a current sheet excitation, providing an incident wave on the nonsample side of the aperture. By utilizing a source-field relationship with the aperture model the incident electric field may be calculated to be[93, 94]:

$$\overline{E}^{i} = \int_{aperture} \left[-\hat{x}\left(2E_{ax} \frac{dg}{dy} \right) + 2\hat{y}\left(E_{az} \frac{dg}{dz} + E_{ax} \frac{dg}{dx} \right) - \hat{z}\left(2E_{az} \frac{dg}{dy} \right) \right] ds \qquad \textbf{5.1}$$

where E_{ax} and E_{az} are the x- and z-directed components of the electric field in the aperture and g is the time-harmonic free-space Green's function.

$$g = \frac{e^{-jkR}}{4\pi R} \qquad \textbf{5.2}$$

where the time convention is e^{jwt} and k is the wavenumber, and where

$$R = \sqrt{(x - x')^2 + (y - y')^2 + (z - z')^2} \qquad \textbf{5.3}$$

After simplification of Equation 5.1, a constant electric field in the aperture produces an incident electric field defined as:

$$\overline{E}^{i} = \hat{z} 2\pi a^2 E_{az} (y - y') \frac{1 + jkR}{4\pi R^3} e^{-jkR} \qquad \textbf{5.4}$$

This model indicates that the detected power is largest when the source aperture is directly behind the wire. This near-field focusing effect is only seen when imaging very small objects such at small wires. It is important to note that the coupling impedance between the aperture and the wire varies as the source moves along the wire and yields a rounded shaped image[93].

Once the NSOM was calibrated with the wire grids, a comparison between a Perkin-Elmer Spotlight model 300 containing a globar source and focal plane array with a 6.25 μm by 6.25 μm pixel size and the NSOM was completed. The corner of the supporting structure of the calibration grid was scanned for comparison of the chemical density map to verify that the NSOM greatly exceeds the resolution of a microspectrometer. Finally, data collection across the cak9 control mouse artery wall with both a 1 mm polystyrene filter in place and a 1 mm water filter in place was completed. Polystyrene and water were chosen because their transmission properties are nearly orthogonal to each other. In other words, a positive response with one should yield a negative response with the other. These scans were completed as a proof of concept, two years before spatial and signal-to-noise optimizations to the NIR-NSOM were completed.

Results and Discussion

The scan conducted of the SEM calibration grid with the NIR-NSOM yielded the results shown in Figure 5.17. It is evident from the figure that three adjacent and parallel calibration wires were seen. The wires are located at approximately 250 nm, 500 nm, and 1000 nm. Although these wires are not evenly spaced, they are representative of the calibration grid. The overall time that this scan took to acquire was approximately 6 hours. During this time, small temperature fluctuations were likely to occur within the NSOM box. A temperature fluctuation of 0.25° C would cause 250 nm of drift at the piezo stage. Such small temperature changes are likely to occur when a highly temperature controlled atmosphere is unavailable. The sole purpose of the calibration is

to determine the spatial resolution of the NIR-NSOM. In this test, three parallel wires 50 nm in diameter were clearly identified with light in the NIR range. Therefore, the NIR-NSOM is able to resolve structures 50 nm and larger.

The two dimensional scans of the corner of the calibration grid supporting structure with the Spotlight and the NIR-NSOM may be compared to determine the improvement of the spatial resolution. The chemical density map from the Spotlight (Figure 5.18) has a spatial resolution that is at least two orders of magnitude worse than the spatial resolution in the chemical density map collected with the NIR-NSOM (Figure 5.19). Since the Spotlight has a spatial resolution of 6.25 μm and the NIR-NSOM has at least a two order of magnitude increase in spatial resolution, it may be inferred that the NIR-NSOM could resolve and structure larger than 62.5 nm.

The results of the one dimensional scan across the cak9 control mouse artery wall with a 1 mm polystyrene filter in place are shown in Figure 5.20. The results of the one dimensional scan across the cak9 control mouse artery wall with a 1 mm water filter in place are shown in Figure 5.21. In both figures the scan without a filter in place is green and the scan with the filter in place is blue. The important aspect here is that the styrene filter gives a positive response when scanning across a vessel wall and the water filter gives a negative response when scanning across the vessel wall. This is a prime example of how molecular factor computing with molecular filters can be tuned to correlate with an analyte of interest. In this case polystyrene correlates to the collagen and elastin in the

normal vessel wall. The orthogonal filter produced the opposite response to collagen and elastin as was expected.

Conclusions

It was expected that NIR-NSOM would exceed the spatial resolution of synchrotron IR-microspectrometry by an order of magnitude or more. Once the NIR-NSOM had been fully optimized and shielded it yielded a spatial resolution at least two orders of magnitude better than traditional microspectrometry. The NIR-NSOM was also able to gather spectra with spatial resolution substantially less than the spectral limits of NIR light. It is important to note that the synchrotron source significantly increased the signal to noise ratio, and by using MFC, NSOM scan time was reduced to a simple raster scan. Since the finely tuned monochromators or interferometers are not used and molecular filters are used instead, MFC-NSOM does not have to wait for long scans at each spatial pixel. Scanning times with as low as 0.1 seconds per pixel have been successfully used. Now, with the addition of molecular filters, simple raster scanning of the surface gives the chemical density plot of the analyte corresponding to that molecular filter. The reduction in data acquisition time has been lowered by several orders of magnitude when compared to the conventional globar source and either scanning interferometers or monochromators.

Chapter Five Figures

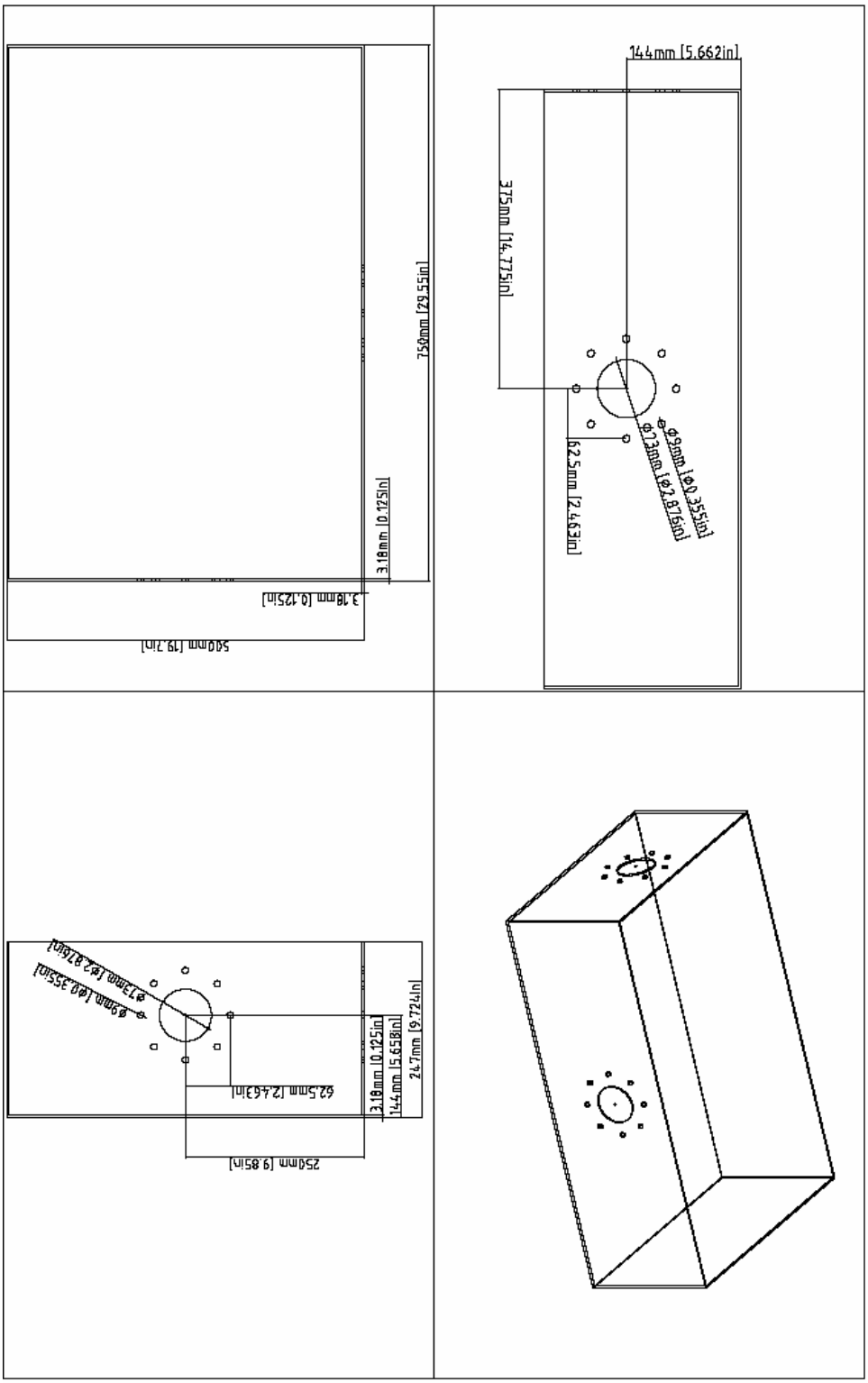

Figure 5.1: Dimensioned aluminum NSOM box schematic.

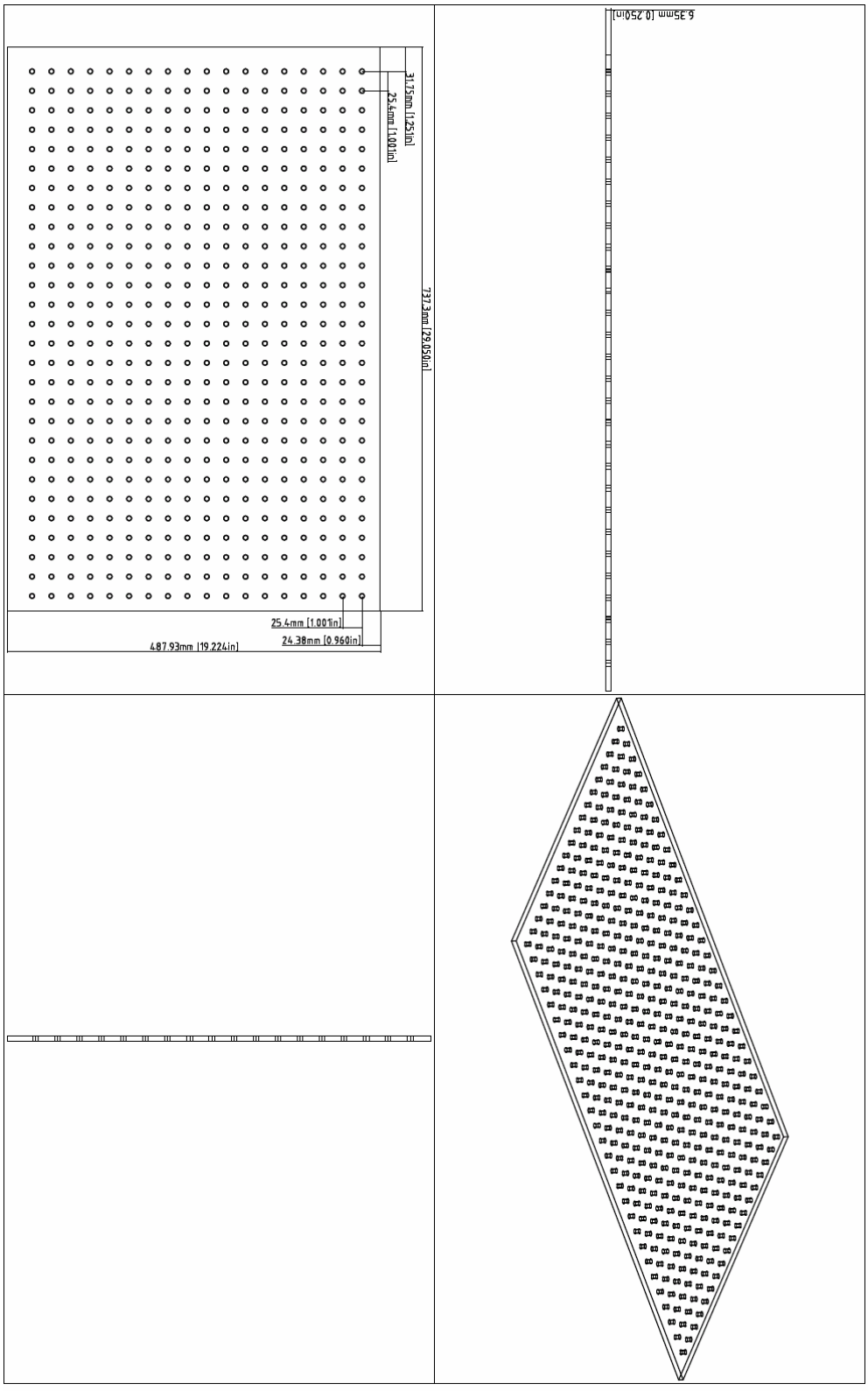

Figure 5.2: Dimensioned aluminum NSOM box false bottom optical mount schematic.

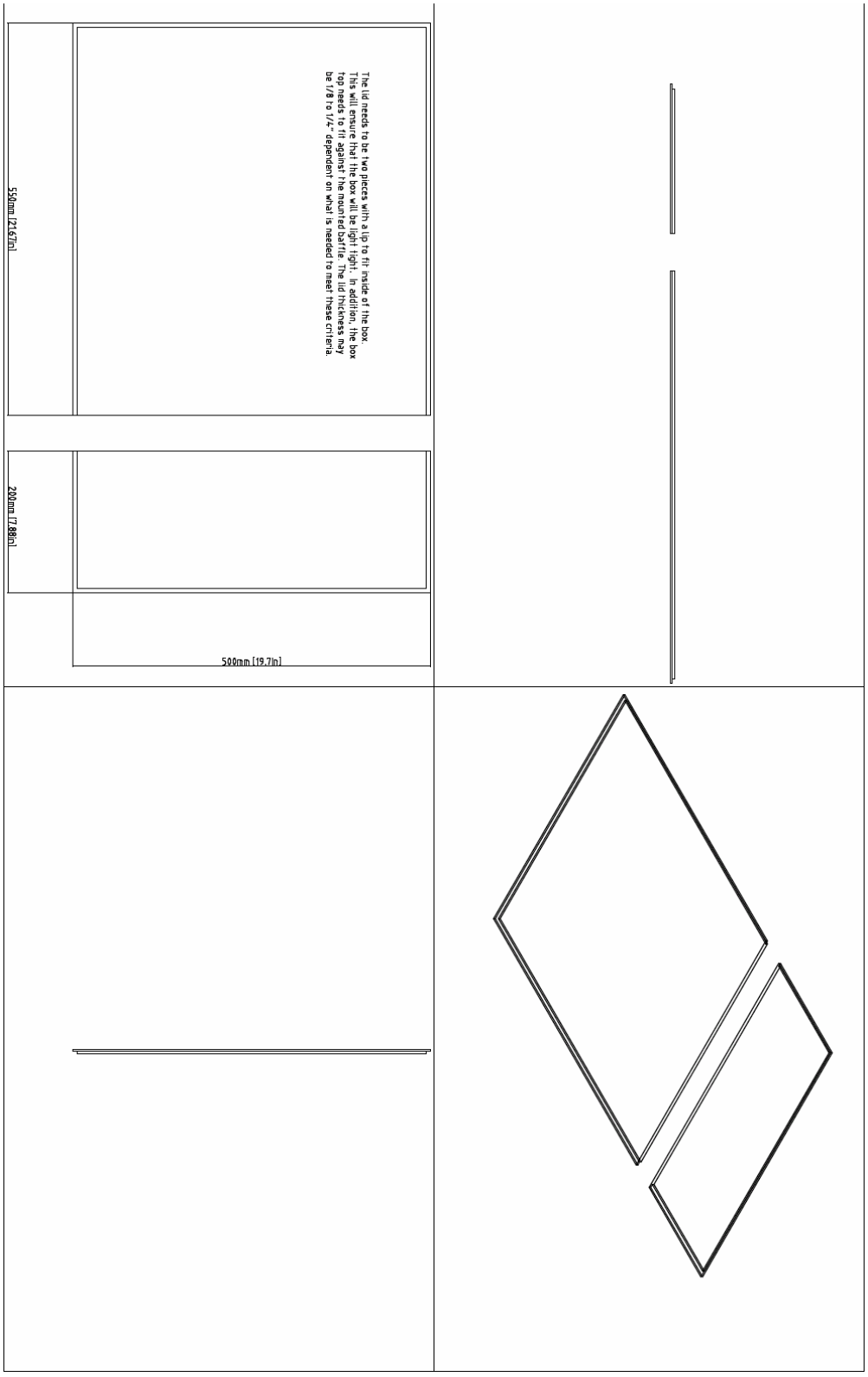

Figure 5.3: Dimensioned aluminum NSOM box lid schematic.

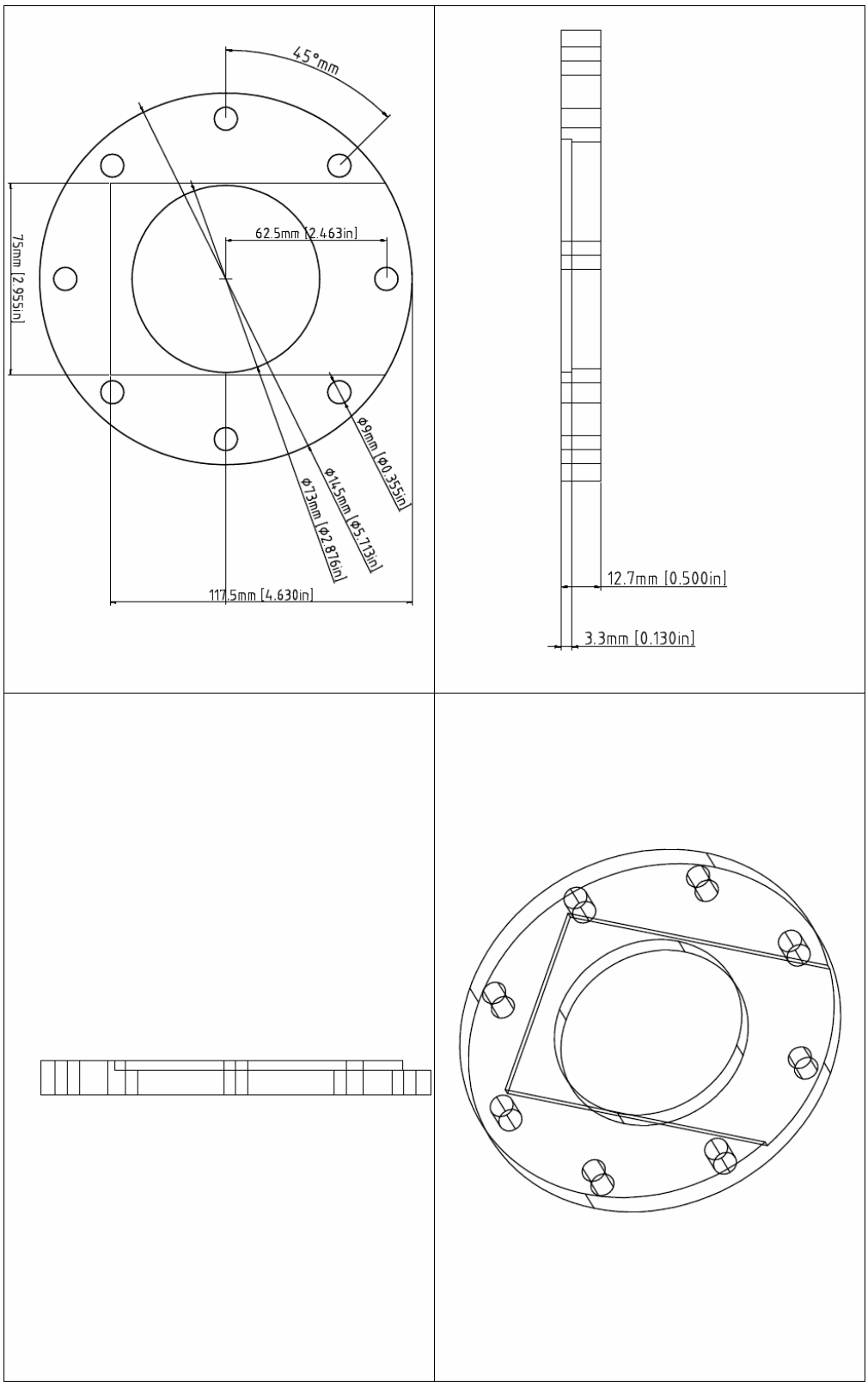

Figure 5.4: Dimensioned aluminum NSOM box shutter adapter schematic.

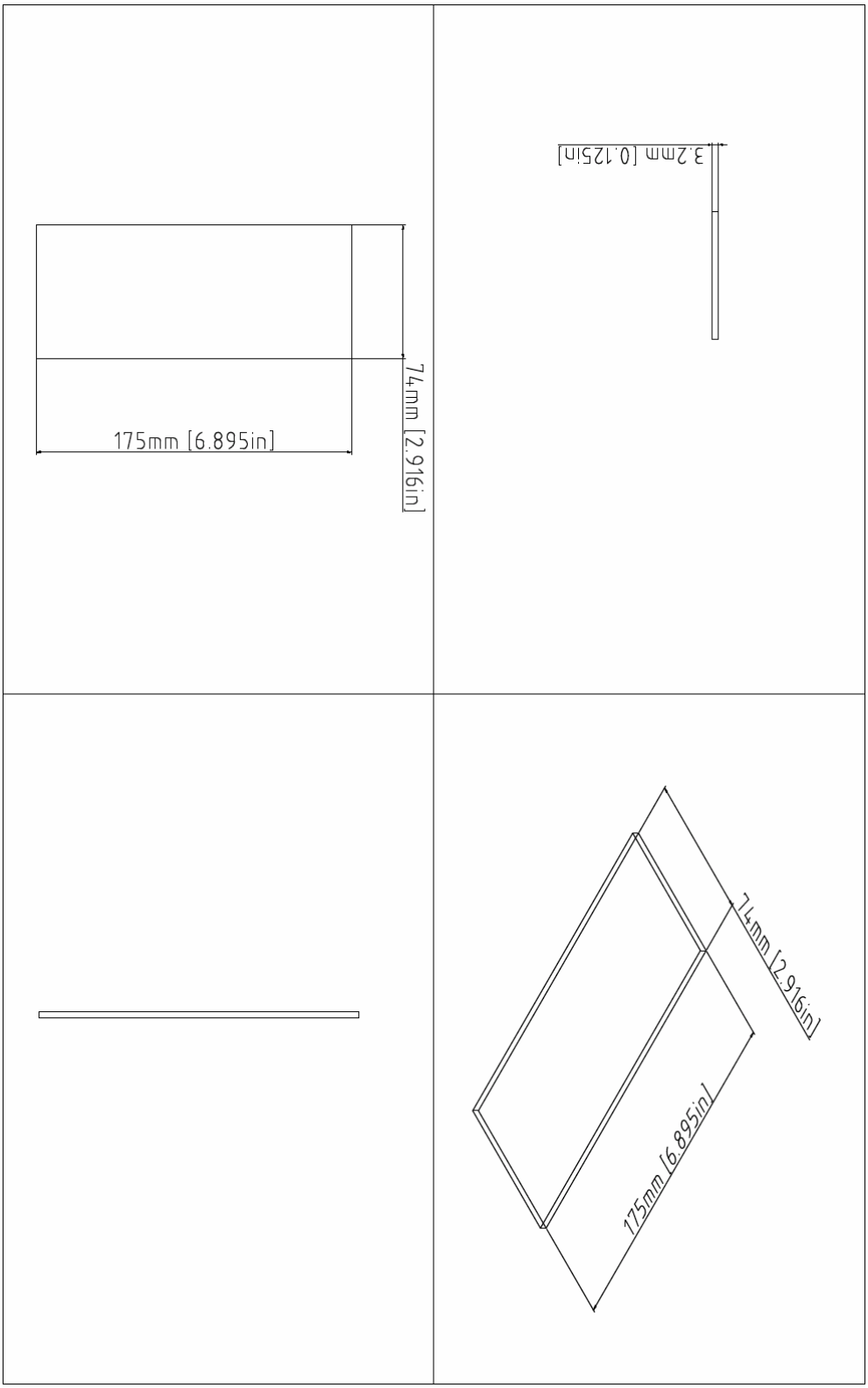

Figure 5.5: Dimensioned aluminum NSOM box shutter schematic.

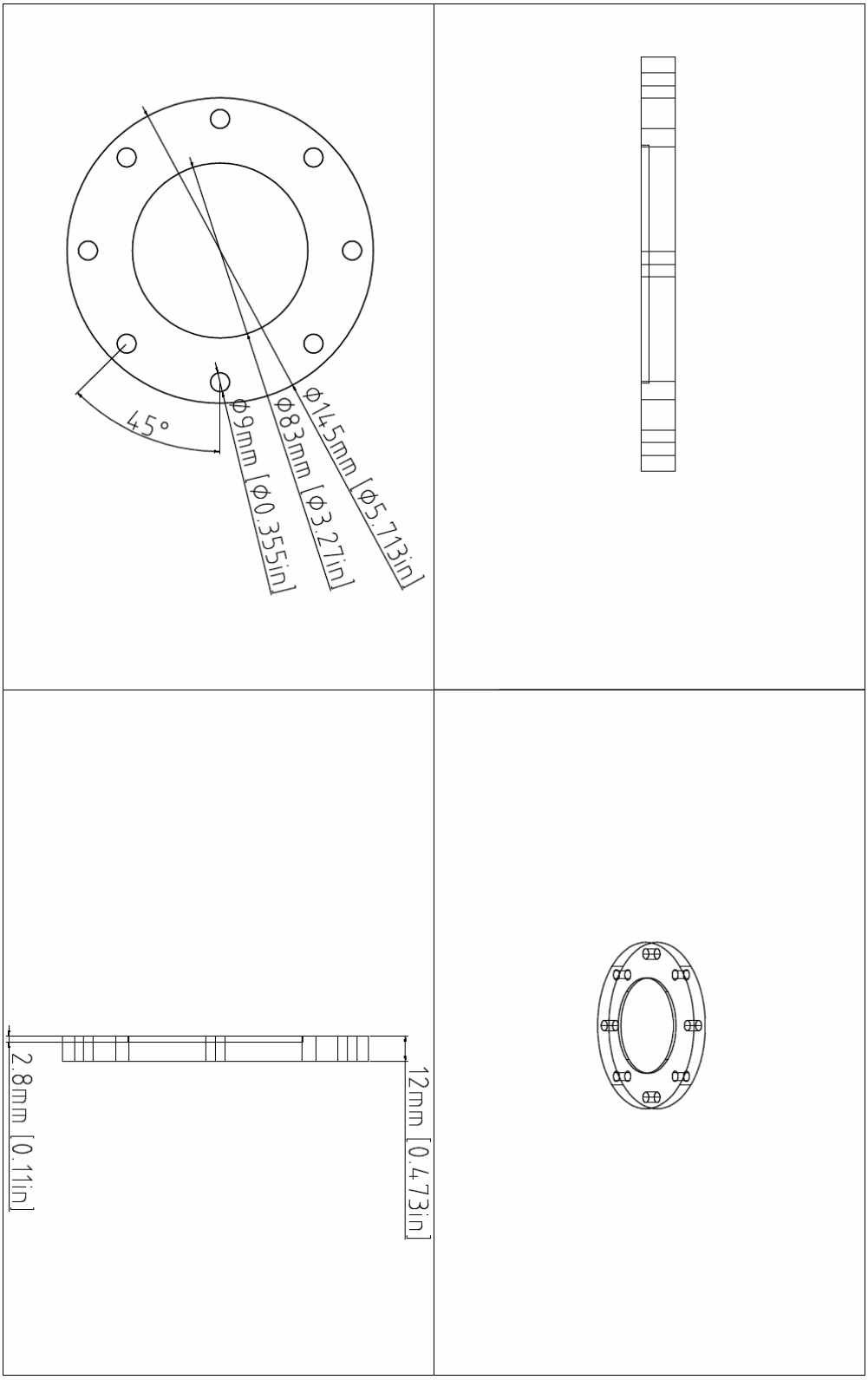

Figure 5.6: Dimensioned aluminum NSOM box end cap schematic.

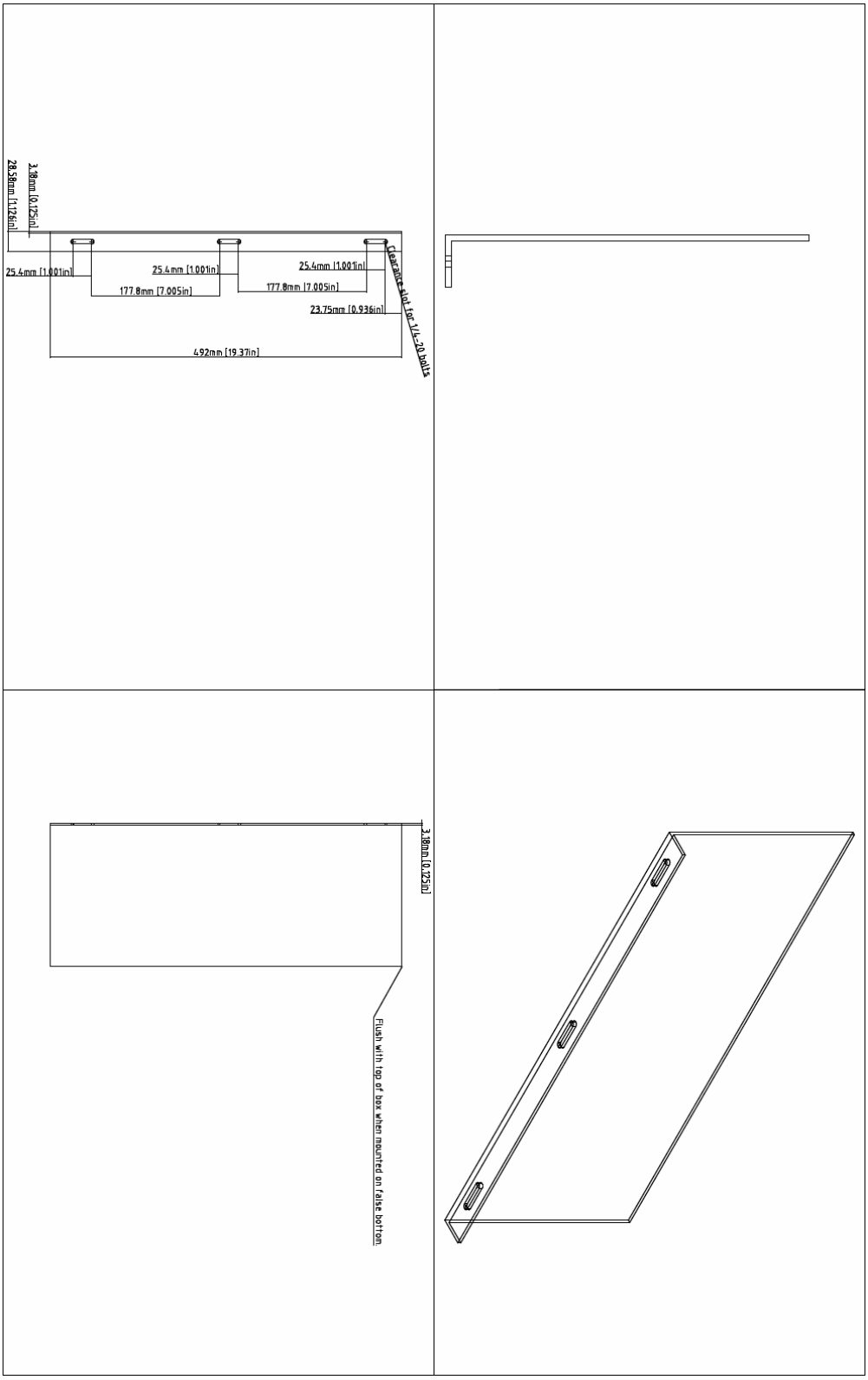

Figure 5.7: Dimensioned aluminum NSOM box baffle schematic.

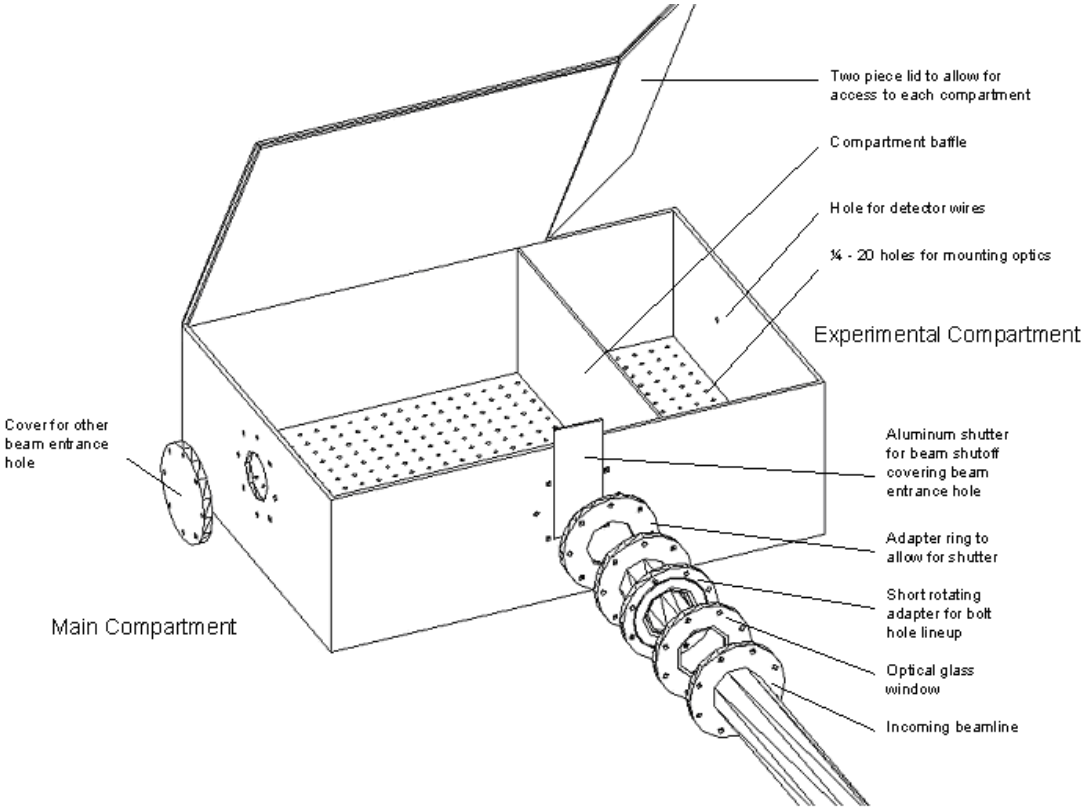

Figure 5.8: Aluminum NSOM box for joining with the synchrotron beam.

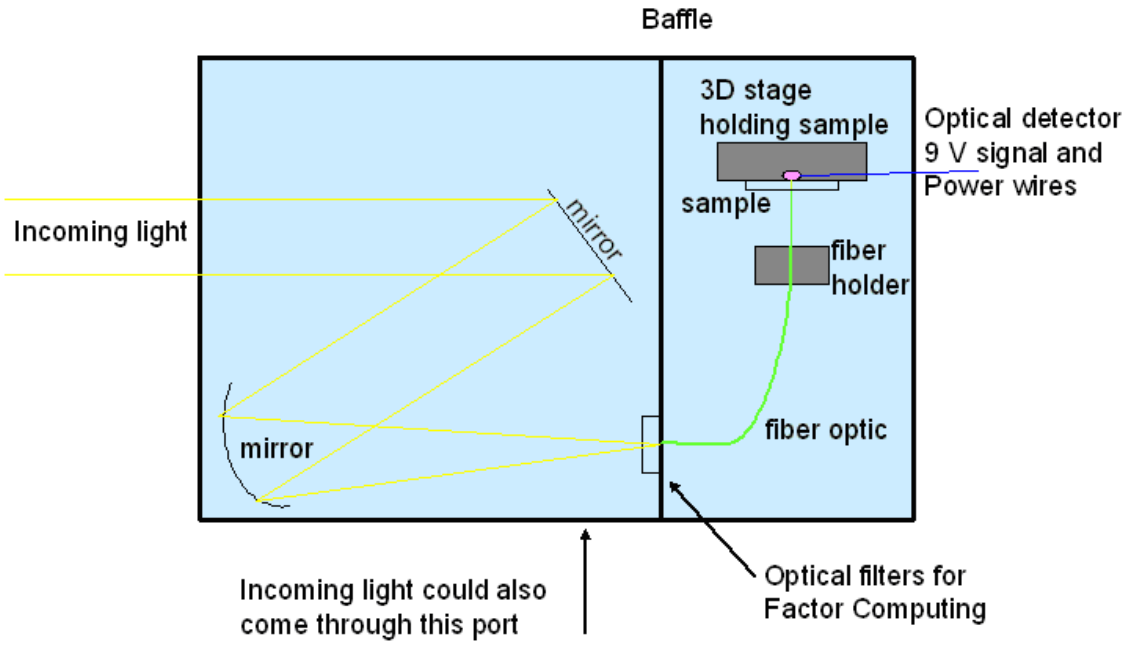

Figure 5.9: Optical schematic for inside the NSOM box.

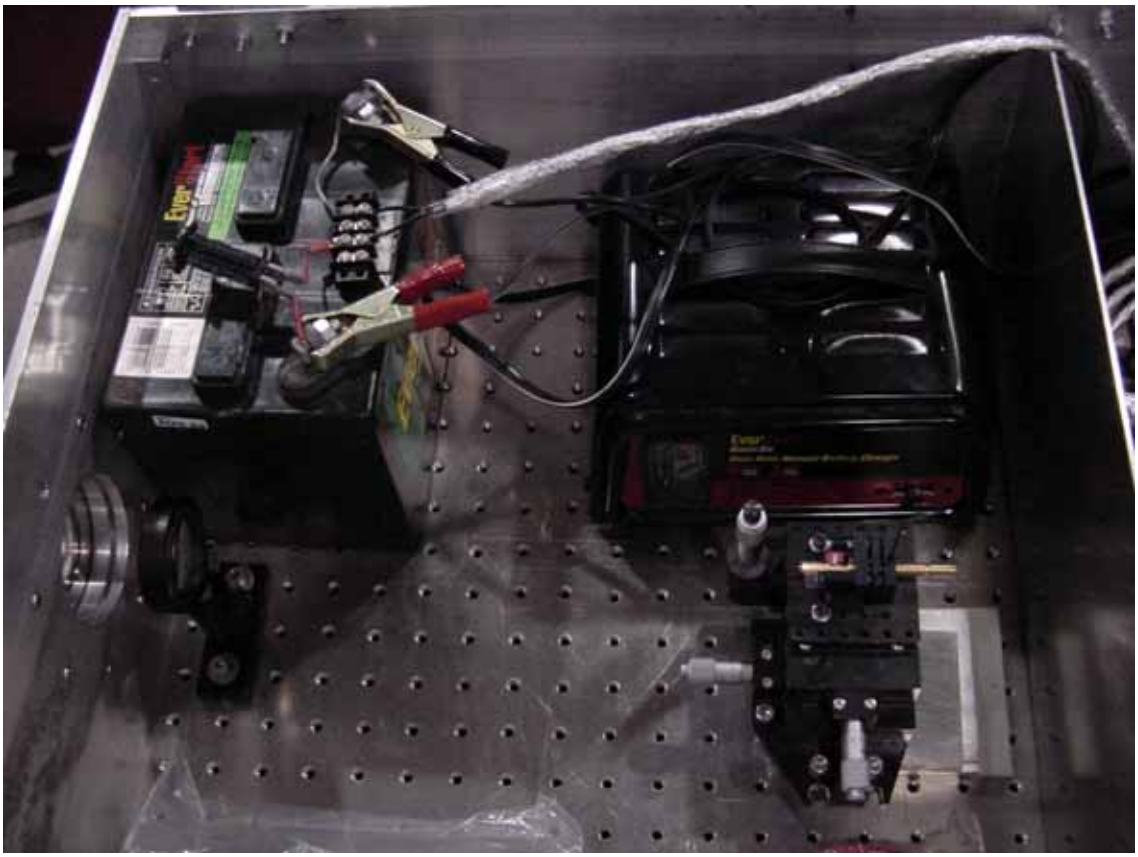

Figure 5.10: Optical setup on the light coupling side of the NSOM box.

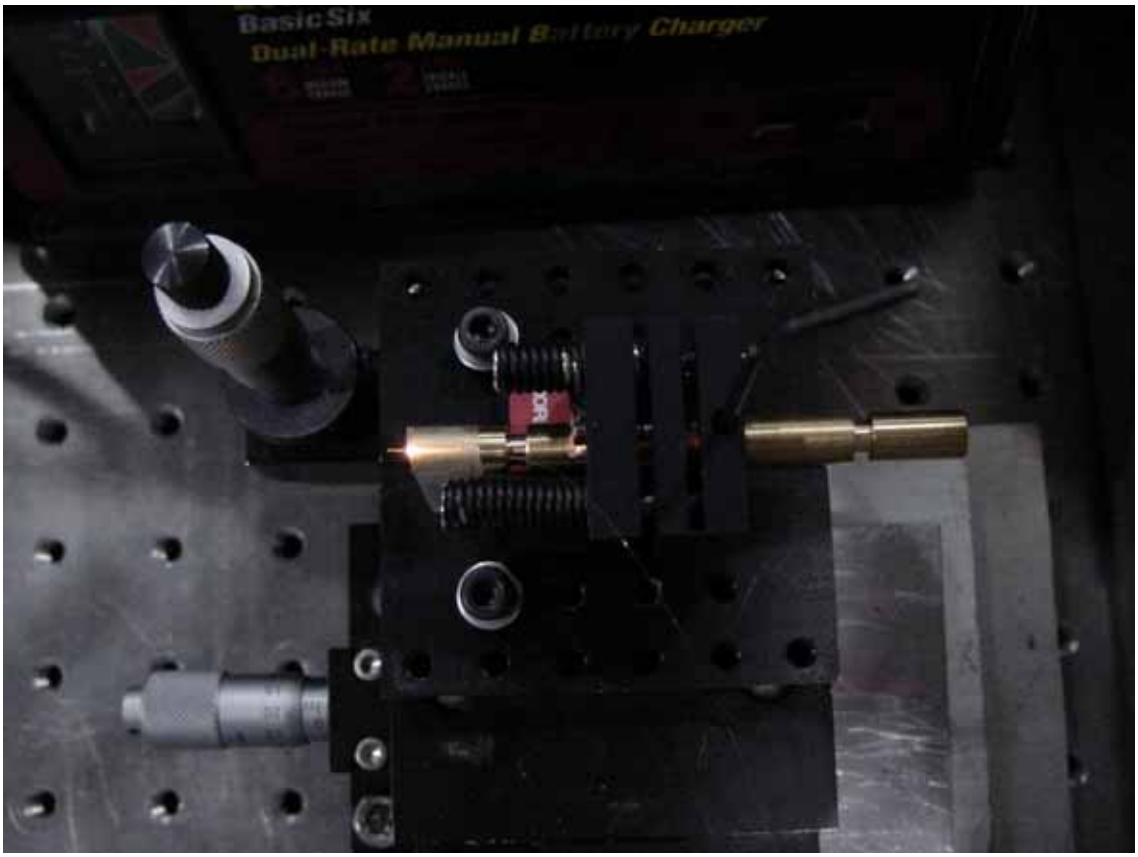

Figure 5.11: Light coupled into the fiber with fine tuning in the x, y, and z directions as well as pitch and yaw at the fiber chuck mount.

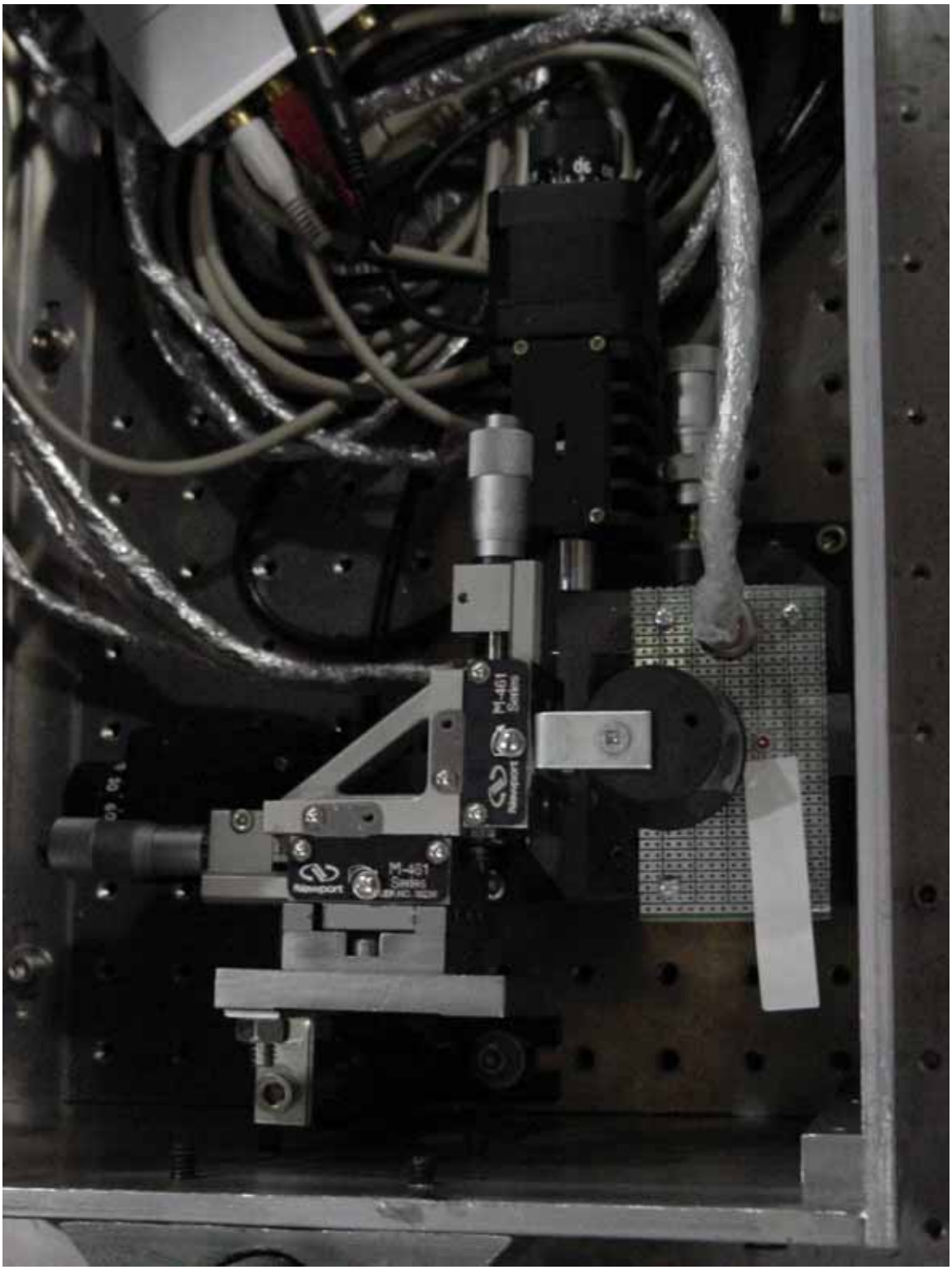

Figure 5.12: Hardware located on the detection side of the NSOM box.

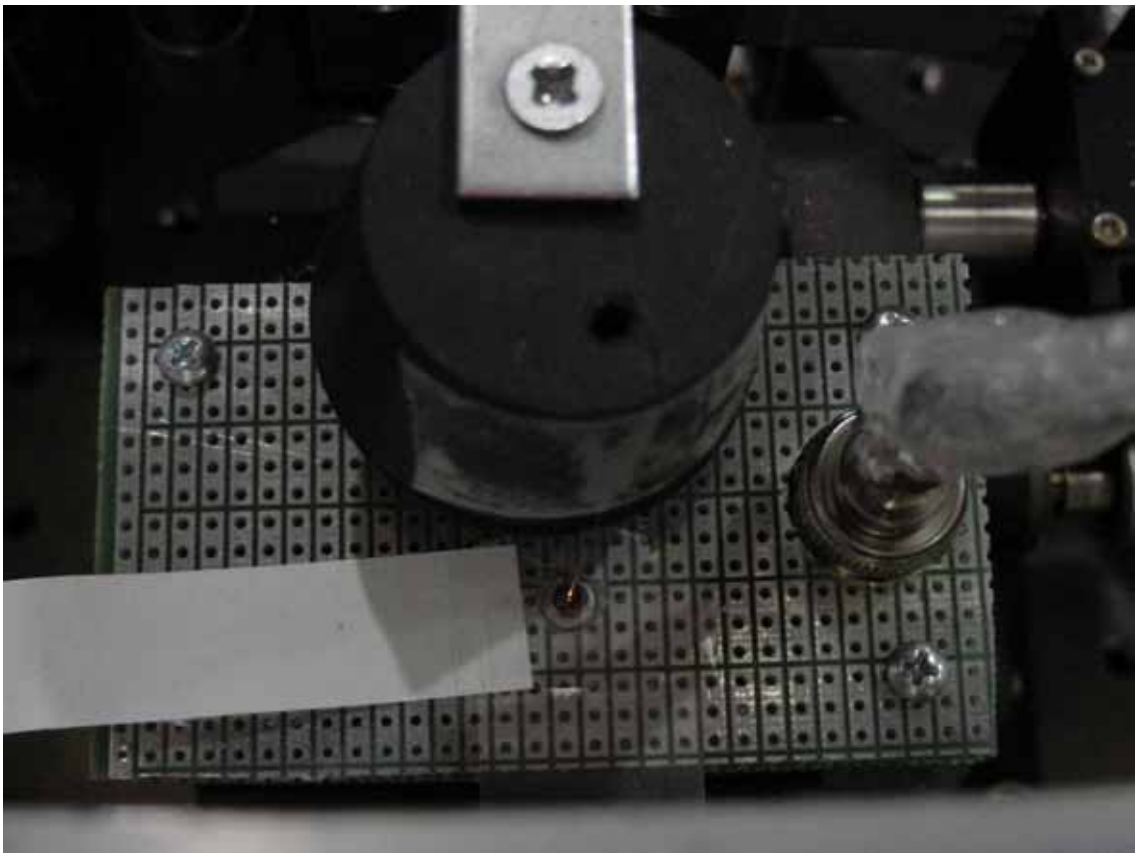

Figure 5.13: Fiber, Sample, and detector located on the detection side of the NSOM box.

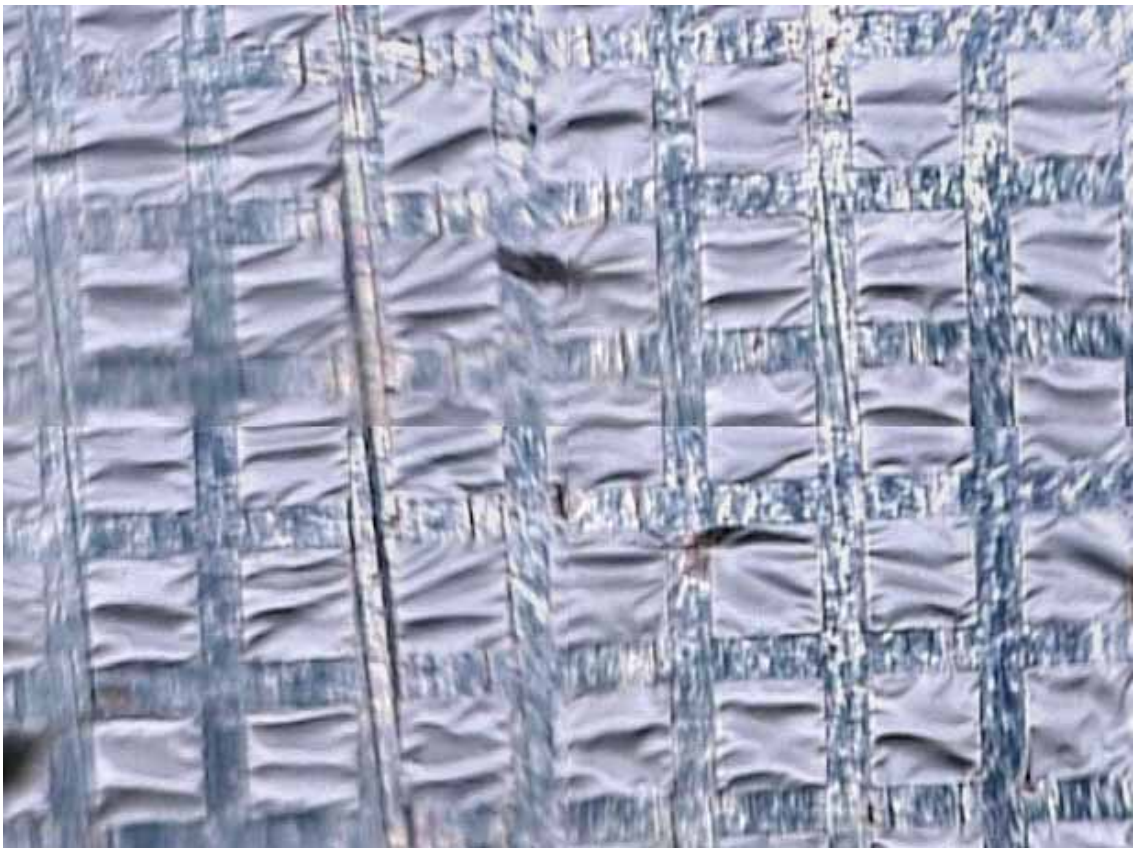

Figure 5.14: A visible image of a calibration grid (grey coating) and supporting structure (darker mesh).

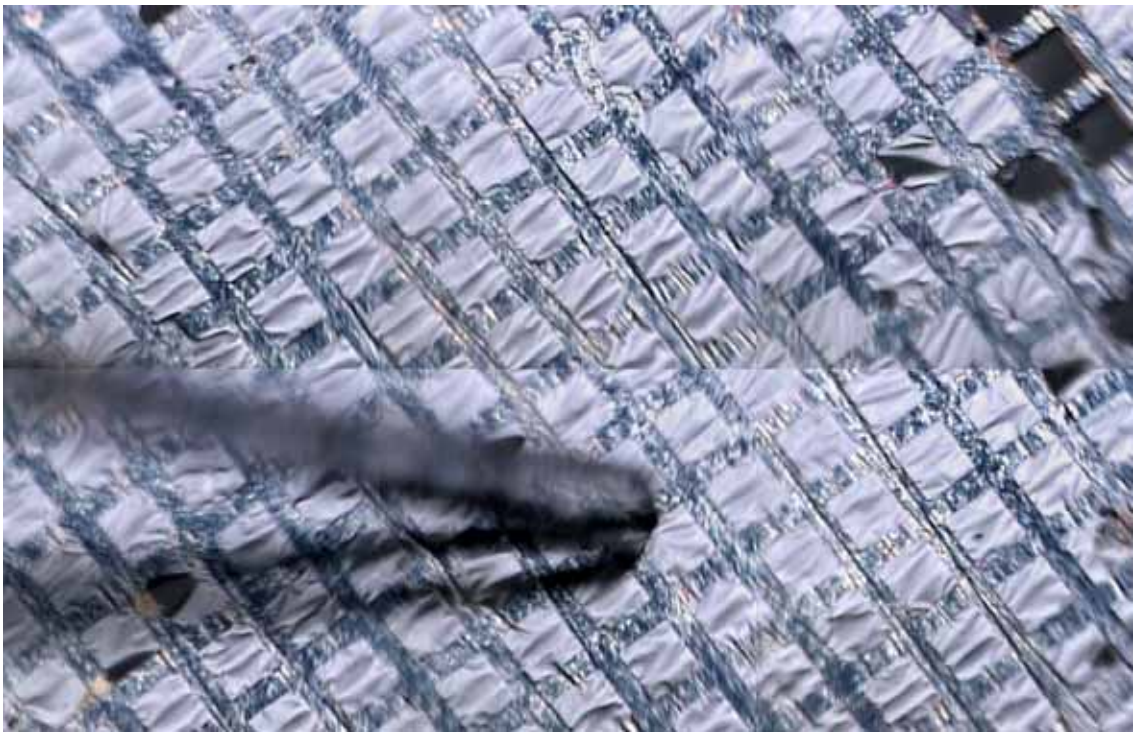

Figure 5.15: A visible image of a calibration grid (grey coating), supporting structure (black mesh), and unaltered 50 μm diameter fiber.

Figure 5.16: A visible image of a calibration grid (grey coating), supporting structure (black mesh), and broken pulled fiber tip.

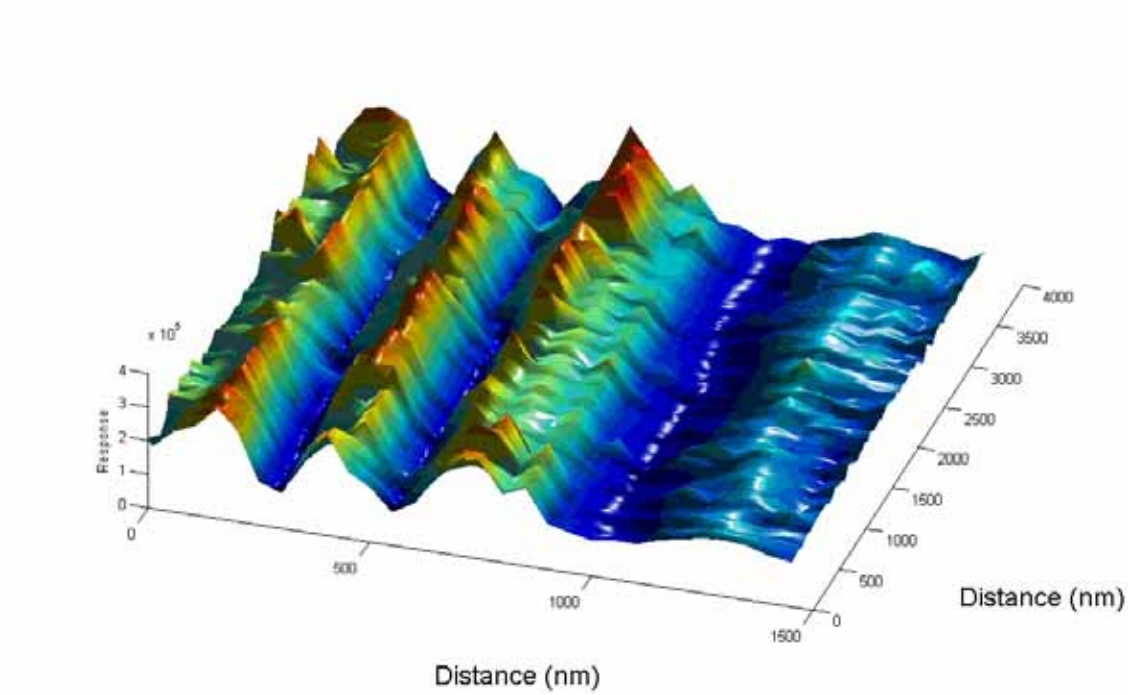

Figure 5.17: NSOM scan of SEM calibration grid in 20 nm increments with data collection at 0.1 s/pixel.

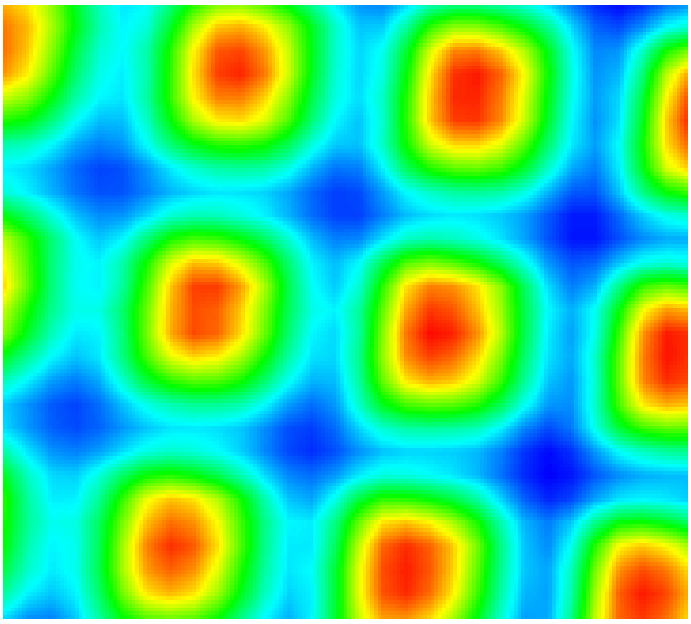

Figure 5.18: Spotlight scan of the SEM calibration grid supporting structure with a 6.25 µm by 6.25 µm pixel size (Centers are separated by ~75 µm).

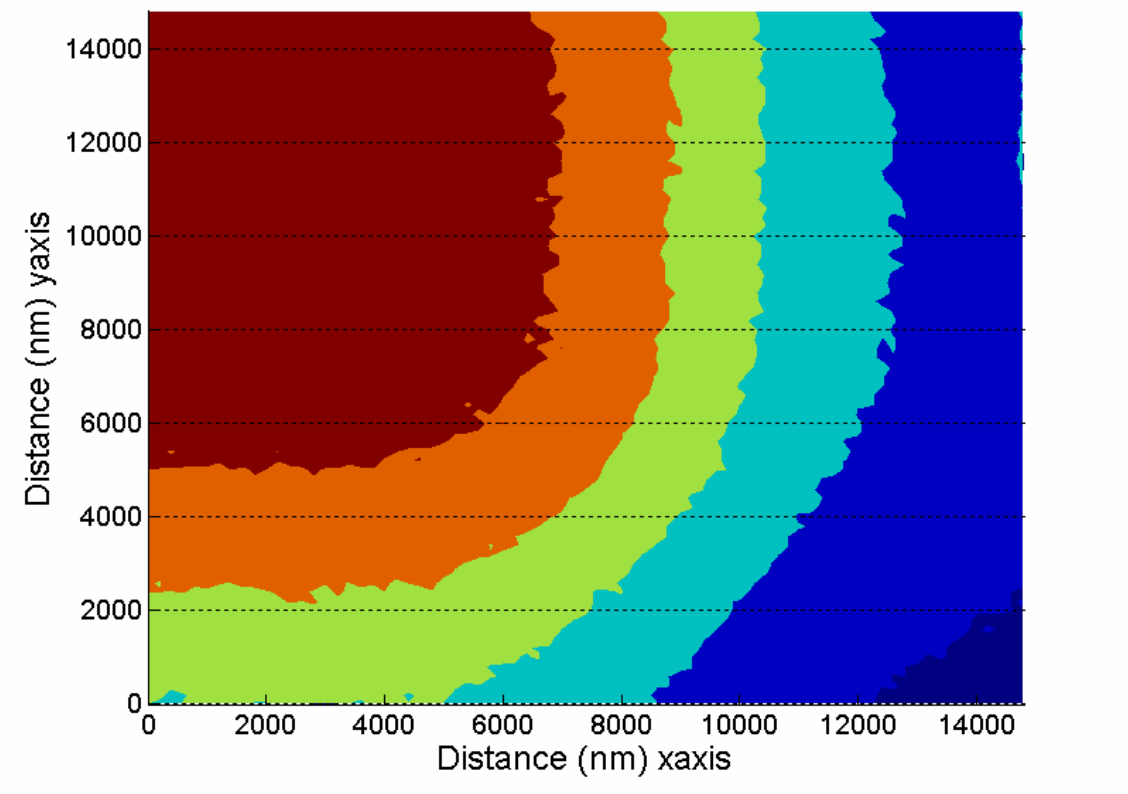

Figure 5.19: NSOM scan of the SEM calibration grid supporting structure with a 20 nm by 20 nm pixel size.

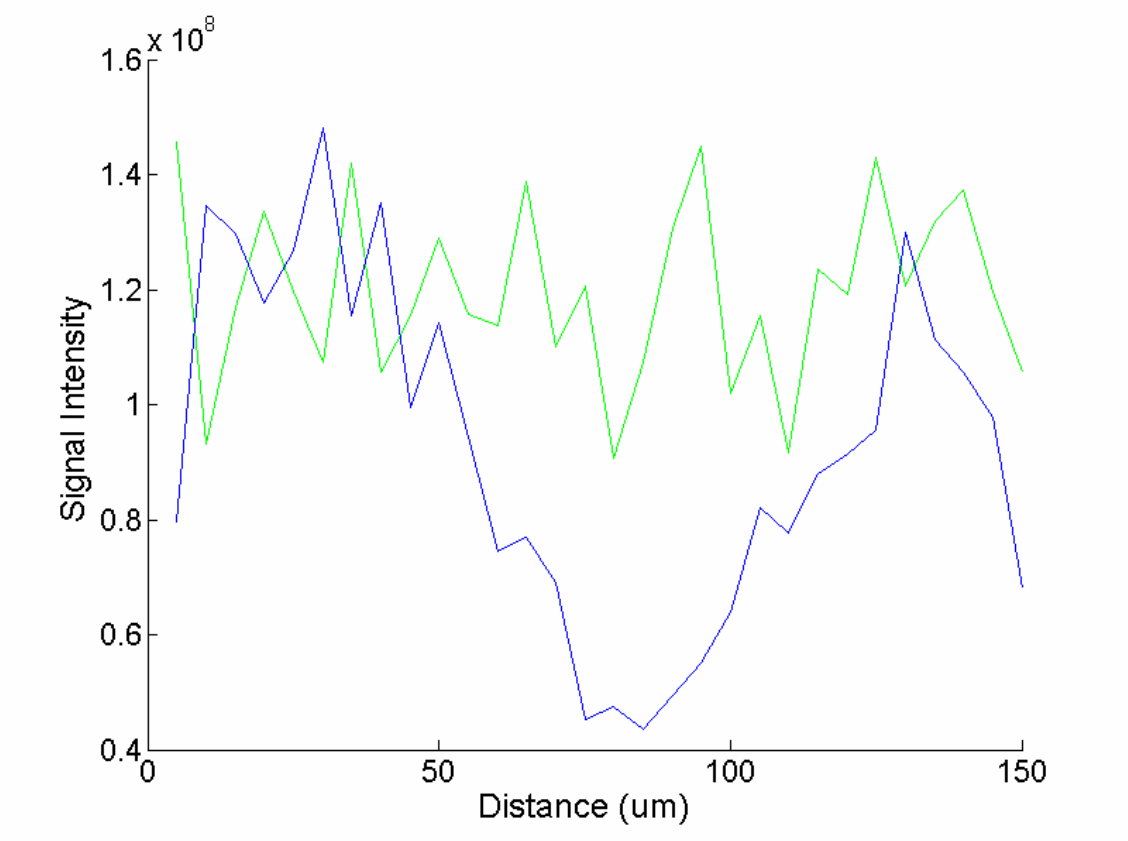

Figure 5.20: NIR-NSOM scan across the cak9 control mouse artery wall with (green line) and without (blue line) a 1 mm styrene filter in place.

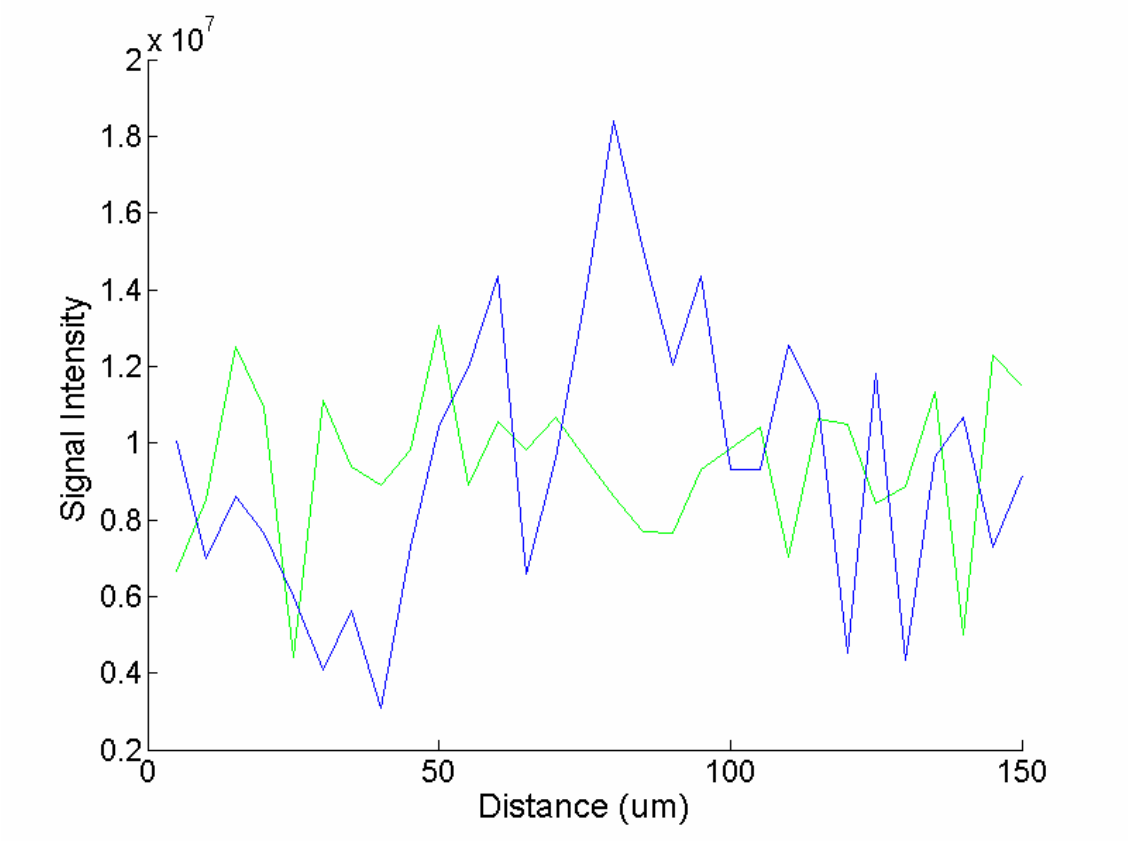

Figure 5.21: NIR-NSOM scan across the cak9 control mouse artery wall with (green line) and without (blue line) a 1 mm water filter in place.

Conclusion of Dissertation

From the work contain herein, it is apparent that new chemometric data analysis techniques combined with novel instrumentation may be successfully applied to a variety of analytical problems. The usefulness of such techniques is determined by the requirements of society as a whole. The limitations of modern computer processing combined with the extensive quantities of data acquired from hyperspectral imaging require new data analysis techniques. The development of SAQQ with its ability to rapidly analyze massive datasets as well as produce digital images based upon proximity to multidimensional population clusters satisfies these requirements. In this dissertation, the potential of SAQQ to be used in various spectroscopic imaging situations from in vivo analysis of disease to the digital staining of tissue samples to spectral analysis in space has been demonstrated. Specifically, the identification of chemical populations with SAQQ followed by their digitally stained images was demonstrated in the murine samples of abdominal aortic aneurysm.

The costs and instrumental requirements for the exploration of extreme environments require new spectrometers that meet the demands imposed. In this dissertation, the SSSI meets many requirements for remote hyperspectral imaging in space. The most important requirement that was demonstrated was the ability to construct a spectrometer with no moving parts that could be used in a commercially made form on a Mars rover. It is an inexpensive spectrometer that could be used in high-school classrooms for teaching the fundamentals of molecular spectroscopy as well. The SSSI combined with

SAQQ creates a useful system for hyperspectral imaging and analysis in extreme environments.

Finally, the demand for a cure to serious illnesses has led to the search and discovery of biochemical pathways leading to disease. New instrumentation that is able to record chemical processes on the sub-cellular level in real time is needed to push forward what is known. The application of the ISP technique, MFC, to NIR-NSOM allows for real time imaging of the processes in individual mitochondria. The theory of MFC combined with NSOM has been demonstrated in this dissertation. In addition, the NIR-NSOM has been coupled with the synchrotron at the NSLS at BNL for future progress in this area.

ISP in general has a tremendous future. Already ISP is being implemented for quantitative and qualitative measurements in acoustic resonance spectroscopy and near-infrared spectroscopy. Optical ISP could be implemented with a series of orthogonal wedge filters that go from 0% to 100% transmittance. This would alleviate the demand on the computer and take all of the principal component analysis to the instrument itself. Time varying pulse sequences could be implemented on the SSSI as another implementation of optical ISP. Fourier Transform SSSI and acoustic resonance spectroscopy are also areas for new development. Finally, ISP could be applied to other difficult spectroscopic problems. ISP-Fluorescence spectroscopy is one of many examples of fields completely untouched by ISP.

Appendix A – MatLab Functions

Contents

Spectral Analysis of Quantile Quantile Plots Functions

Function A1 - loaddata

```
%Copyright 2005 Justin Clay Harris
%the function called is loaddata, and it has the form:
%loaddata('begintext',beginnum,endnum,'endtext').
%begintext = is the text string at the beginning of the filename
%beginnum = is the starting number in the file's filename
%endnum = is the ending number in the file's filename
%endtext = the ending text string of the filename including .CSV


function loaddata(begintext,beginnum,endnum,endtext)

predata = importdata ([begintext,num2str(beginnum),endtext]);
data = zeros((length(predata)),(endnum - beginnum + 1));
wavelengths = predata(:,1);

m = 1;
tic
%loops through the files to be loaded into data
for i = 1:1:(endnum - beginnum + 1),
   predata = importdata ([begintext,num2str(i + beginnum - 1),endtext]);
   data (:,i) = predata(:,2);

   if 1 == (i/1000)/m,
      m
      toc
      m = m + 1;

   end

end

data = data';

save (begintext,'data','wavelengths');
```

Function A2 - prepdata

```
function h = prepdata (data)

if min(min(data)) >= 0 && max(max(data)) >= 20,
   if min(min(data)) == 0,
      data = (data + 0.0001);
   end
   data = log(100./data);
end

if min(min(data)) <= 0,
   data = (data - min(min(data)) + 0.0001);
end

data = mscorrect(data);
data = zscore(data);

assignin('base','zabsdata',data)



end
```

Function A3 - SAQQ

```
function h =
saqq(data,wavelengths,loads,s,scores,scorenumber,optr2,percentpoints,normal)
%   function h =
saqq(data,wavelengths,loads,s,scores,scorenumber,optr2,percentpoints,normal)
%
%   where:
%      data is the original data set.
%
%      wavelengths is an array of the collection wavelengths
%
%      loads is the loadings from optPCA.
%
%      s is the s from optPCA.
%
%      scores is the array of scores from spectralPCA or any other
%            generic pca.  Each set of scores is in each column.
%
%      scorenumber is the number of the column in scores from which the
%            quantile-quantile plot is made.
%
%      optr2 is the minimum r-sqared value allowed for a line through the
%            data of the quantile-quantile plot.  This value is used to
%            determine the endpoint of each cluster.  If there are too
%            many clusters in the result, decrease rsqr.  If there are
%            too few clusters, increase rsqr.
%               *** DEFAULT optr2 = 0.9 ***
%
%     percentpoints is a value from 0 to 100 percent that represents the
%            minimum length of each line segment.  As a result, this
%            defines the minimum size of each cluster.
%               *** DEFAULT percentpoints = 10 ***
%
%      normal is a column of data representing a normal distribution for
%            plotting scores against.  If this is not specified, a
%            standard normal will be used. The number of rows in normal
%            must equal the number of rows in scores(:,scorenumber).
%            *** WARNING - discontinuities in this data set
%            will result in erroneous results. ***
%               *** DEFAULT "normal" is a standard normal distribution ***
%
%
%
%   saqq displays an empirical quantile-quantile plot and then
```

```
%   estimates clusters from the quantile-quantile plot.  To do this
%   saqq uses the r^2 value from linear regression to isolate
%   linear portions of the plot as well as checking for discontinuities in
%   the verticle direction.  The starting point for finding the first
%   cluster is defined as the left most point on the quantile-quantile
%   plot, which is also the minimum.  The first endpoint for linear
%   regression is defined as the begining point plus the number of points
%   times percentpoints or nxend = (nxbeg + (round(nx*(percentpoints/100)))).
%   The resulting r-squared value is checked against the predetermined rsqr
%   value.  If the result is larger than rsqr, then nxend is incremented by
%   the predetermined increment size.  Once resultant r-squrare drops below
%   the input rsqr, then the loop is halted and the beginning and endpoints
%   are reset.  The beginning point of the next cluster becomes the end
%   point plus one and the endpoint again becomes the begining point plus
%   the number of points times percentpoints or nxbeg = nxend + 1 and
%   nxend = (nxbeg + (round(nx*(percentpoints)))).  This continues until
%   the end point equals the number of points.
%
%
%
%   QQPLOT(scores,scorenumber) makes an empirical QQ-plot of
%   the quantiles of the data set scores(normal) versus the standard
%   normal.
%
%   QQPLOT(scores,scorenumber,normal) makes an empirical QQ-plot of
%   the quantiles of the data set scores(scorenumber1) versus the quantiles
%   of the data set normal.  This plot assumes that
%   scores normal is normal, if scores(normal) is not normal
%   then this function will likely give erroneous results.
%
%
%   The default quantiles are those of the smaller data set.
%
%
%   The purpose of the quantile-quantile plot is to determine whether
%   the sample in scorenumber1 is drawn from a Normal (i.e., Gaussian)
%   distribution, or whether the samples in scorenumber1 and normal
%   come from the same distribution type.  If the samples do come from the
%   same distribution (same shape), even if one distribution is shifted
%   and re-scaled from the other (different location and scale parameters),
%   the plot will be linear.
%
%   Heavily Edited version by Justin C. Harris from MatLab's routine...
%   Copyright 1993-2004 The MathWorks, Inc.
%   $Revision: 2.13.2.1 $  $Date: 2004/01/24 09:36:41 $
```

```
width = 61;  %image width in pixels
height = 118;  %image height in pixels

% if width*height ~= length(data),
%     error('The value input for length or width is not correct based upon the number of
spectra in the input dataset.');
% end

tic;
close all hidden    %   Closes all figures before the the program is run
pause (0.01);
fprintf('Calculating Quantile-Quantile Plots and QQ Plot Linear Regression.\n')

%   Checks to see if any absorbtion values are below zero and adds a
%   baseline correction if there are values below zero.

minscores = min(min(scores));   %   Finds the minimum value of scores

if minscores <= 0,  %   If the minimum value of scores is less than or equal to zero...
   scores = (scores - minscores + 0.0001); %   Add a baseline correction so the lowest
value is 0.0001.
end

if nargin <= 5 | nargin >= 10,   %   This checks to make sure that the number of inputs is
appropriate
error('Input must contain four to seven parameters.');   %   Otherwise it responds with an
error

%   If the number of inputs is 6, 7, or 8, then scores(scorenumber1) will
%   be plotted versus the standard normal.  optr2 and percentpoints values
%   will be set to their defaults if these values have not been defined.

elseif nargin >= 6 && nargin <=8,

   x = scores(:,scorenumber);  %   x is set from the input parameters to the appropriate
column of data
   y  =  sort(x);  %   y is set to the sorted values of x

%    PLOTPOS Computes plotting positions for a probability plot
%
%    PP = PLOTPOS(SX) computes the plotting positions for a probabilty
%    plot of the columns of SX (or for SX itself if it is a vector).
%    SX must be sorted before being passed into PLOTPOS.  The ith
%    value of SX has plotting position (i-0.5)/n, where n is
```

```
%     the number of rows of SX.  NaN values are removed before
%     computing the plotting positions.

   x  = plotpos(y);

%     NORMINV Inverse of the normal cumulative distribution function (cdf).
%
%     X = NORMINV(P,MU,SIGMA) returns the inverse cdf for the normal
%     distribution with mean MU and standard deviation SIGMA, evaluated at
%     the values in P.  The size of X is the common size of the input
%     arguments.  A scalar input functions as a constant matrix of the same
%     size as the other inputs.
%
%     Default values for MU and SIGMA are 0 and 1, respectively.
%
   x  = norminv(x);
   xx = x; %   Sets xx to the new values of x
   yy = y; %   Sets yy to the new values of y

   nx = size (x,1);    %   sets nx to the number of rows of x
   ny = size (y,1);    %   sets ny to the number of rows of y


%   Sets the defaults for optr2 and percentpoints if they are not given in
%   the input parameters.  This occurs for inputs of 2 or 3 parameters.  If
%   the number of parameters input is 4, then both of these are defined.
%   If the number of parameters input is 5, then it would not fall within
%   this portion of the if statement.


   if nargin == 6,     %  If the number or parameters input is 6, then set optr2 and
percentpoints to their default values
       optr2 = 0.9;    %  Sets the maximum r^2 value the best fit line may have to its
default value
       percentpoints = 10; %   Sets the minimum line segment length to its default (10% of
the total data set)
   elseif nargin == 7,  %  If the number or parameters input is 7, then set percentpoints to
its default value
       percentpoints = 10; %   Sets the minimum line segment length to its default (10% of
the total data set)
   end

%   Otherwise the number of input parameters is 8,  which makes an
%   empirical QQ-plot of the quantiles of the data set scores(scorenumber1)
%   versus the quantiles of the data set normal.
```

```
else
   y = scores(:,scorenumber);  %   y is set from the input parameters to the appropriate
column of data
   x = normal; %   x is set from the input parameters to the appropriate column of data

   nx = size (x,1);    %   sets nx to the number of rows of x
   ny = size (y,1);    %   sets ny to the number of rows of y

end

%   Check for errors in the input data.

if optr2 <= 0 | optr2 >= 1, %   if optr2 is not between 0 and 1 then an error occurs
   error('rsqr must be between 0 and 1');  %   displays error message
end

%   if percentpoints multiplied by the number of points is not between two
%   data points and the total number of points then an error occurs
if ceil(nx*percentpoints/100) <= 2 | ceil(nx*percentpoints/100) >= nx,
   %   displays error message
   error('percentpoints multiplied by the number of points must be between two data
points and the total number of points')
end

%   if the number of rows in the x and y arrays, corresponding to input
%   data scores and normal are not equal then an error occurs
if nx~=ny,
   %   displays error message
   error(strcat(['The number of rows in scores(:,' num2str(scorenumber) ') must equal the
number of rows in normal.']));
end

xx = sort(x);   %   sets xx to the sorted values of x
yy = sort(y);   %   sets yy to the sorted values of y


%outputs the xy values to the workspace as array qqplotdata
qqplotdata(:,1) = (xx); %   sets column 1 of qqplotdata to the xx values
qqplotdata(:,2) = (yy); %   sets column 2 of qqplotdata to the yy values
assignin('base','qqplotdata',qqplotdata)    %   outputs qqplotdata to the workspace
clear qqplotdata


q1x = prctile(x,25);
```

```
q3x = prctile(x,75);
q1y = prctile(y,25);
q3y = prctile(y,75);
qx = [q1x; q3x];
qy = [q1y; q3y];


dx = q3x - q1x;
dy = q3y - q1y;
slope = dy./dx;
centerx = (q1x + q3x)/2;
centery = (q1y + q3y)/2;
maxx = max(x);
minx = min(x);
maxy = centery + slope.*(maxx - centerx);
miny = centery - slope.*(centerx - minx);

mx = [minx; maxx];
my = [miny; maxy];




   decnxend = 1;    %  sets the amount nxend is increased each iteration
   minend = percentpoints/100; %  multiplies by number of scores for setting the
minimum best fit line
   nxbeg = 1;  %   set nxbeg to the beginning of the array
   nxend = 1;  %   set nxend to the beginning of the array
   plotvalues = 0; %   initiates a new variable
   r2 = 1; %   sets the r2 values for the initial while test

while nxend ~= nx + 1,

   if nxend < (nx - (ceil(nx*minend)));
      nxend = nxbeg + (ceil(nx*minend));
   else
      nxend = nx;
   end

   i = nxbeg;

    while  r2 > optr2 && nxend ~= nx + 1,   %   until r^2 reaches this minimum value,
loop
```

```
      if (yy(nxend)-yy(nxend-1)) > (yy(nxend-1)-yy(nxbeg)) , % or % if (yy(nxend)-
yy(nxend-1)) > (yy(nxend-1)-yy(ceil(0.5*nx*minend))) ,
         if plotvalues == 0,
            plotvalues = [nxbeg,(nxend - 1)];    %   create array and output the beginning
and ending values to an array
            nxend = nxend + 1;
         elseif nxend >= nx,
            if nxend == nx,
               plotvalues = [plotvalues;[(nxbeg-1),(nxend)]];
               nxend = nxend + 1;
            else
               plotvalues = [plotvalues;[(nxbeg-1),(nx)]];
               nxend = nx + 1;
            end
         else
            plotvalues = [plotvalues;[(nxbeg-1),(nxend-1)]];   %   otherwise, append
beginning and ending values to the array
            nxend = nxend + 1;
         end
         nxbeg = nxend - 1;
         r2 = 0;
      else,
         xx2 = (xx(nxbeg:nxend));  %   sets the x value for input into (getrsqr)
         yy2 = (yy(nxbeg:nxend));  %   sets the y value for input into (getrsqr)

         r2 = getrsqr(xx2,yy2);  %   calculates bestfit and r^2

         if    r2 > optr2 && nxend ~= nx,  %   if r^2 is less than the minimum r^2
            if nx >= nxend + decnxend,
               nxend = nxend + decnxend; %   shortens the number of points for fitting next
iteration
            else
               nxend = nx;
            end

         elseif plotvalues == 0,
            if r2 <= optr2,
               plotvalues = [nxbeg,(nxend-1)];    %   create array and output the beginning
and ending values to an array
               nxend = nxend + 1;
            else,
               plotvalues = [nxbeg,nxend];
               nxend = nxend + 1;
            end
```

```
        elseif nxend == nx,
            plotvalues = [plotvalues;[(nxbeg-1),(nxend)]];
            nxend = nxend + 1;
        else
            plotvalues = [plotvalues;[(nxbeg-1),(nxend-1)]];   %   otherwise, append
beginning and ending values to the array
            nxend = nxend + 1;
        end
    end
  end

  if nxend ~= nx,
    nxbeg = nxend;  %   nxbeginning is set equal to nxend
    r2 = 1;
  end

  time = (round(toc*10))/10;
  fprintf('Elapsed time is % 5.1f seconds.\n', time)

end



plotvalues


scrsz = get(0,'ScreenSize');
h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
set(axes,'FontSize',14);

hold

plotvaluesrows = size(plotvalues,1);

plotstr = ['b.';'r.';'g.';'c.';'m.';'y.';'k.'];

for i = 1:1:(ceil(plotvaluesrows/7)),
  plotstr = [plotstr ; 'b.';'r.';'g.';'c.';'m.';'y.';'k.'];
end

for i = 1:1:plotvaluesrows,

plot(xx(plotvalues(i,1):plotvalues(i,2)),yy(plotvalues(i,1):plotvalues(i,2)),plotstr(i,:),qx,q
y,'-',mx,my,'-.');
end
```

```
xlabel('X Quantiles','FontSize',18);
ylabel('Y Quantiles','FontSize',18);


title (strcat(['QQ Plot from PC ' (num2str(scorenumber))]),'FontSize',20);

saveas (h,strcat(['QQ Plot from PC ' (num2str(scorenumber)) '.bmp']))

hold off

h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
set(axes,'FontSize',14);
hold

for i = 1:1:plotvaluesrows,

plot(xx(plotvalues(i,1):plotvalues(i,2)),yy(plotvalues(i,1):plotvalues(i,2)),plotstr(i,:),qx,q
y,'-',mx,my,'-.');
   getresid(xx(plotvalues(i,1):plotvalues(i,2)),yy(plotvalues(i,1):plotvalues(i,2)));
end

xlabel('X Quantiles','FontSize',18);
ylabel('Y Quantiles','FontSize',18);
title (strcat(['QQ Plot with Best Fit Lines and Residuals from PC '
(num2str(scorenumber))]),'FontSize',20);

saveas (h,strcat(['QQ Plot with Best Fit Lines and Residuals from PC '
(num2str(scorenumber)) '.bmp']));

hold off

xx = xx2;
yy = yy2;




[scores,scoresindex] = sortrows(scores,scorenumber); % sorts the x score rows to
correlate with the qqplot x output

h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
set(axes,'FontSize',14);
hold
```

```
for i = 1:1:plotvaluesrows,

plot3(scores(plotvalues(i,1):plotvalues(i,2),scorenumber),scores(plotvalues(i,1):plotvalue
s(i,2),(scorenumber+1)),scores(plotvalues(i,1):plotvalues(i,2),(scorenumber+2)),plotstr(i,:
));
   if i == 1,
      means = mean(scores(plotvalues(i,1):plotvalues(i,2),:));
   else,
      means = [means;(mean(scores(plotvalues(i,1):plotvalues(i,2),:)))];
   end
end
assignin('base','means',means)    %   outputs means to the workspace


xlabel (strcat(['Principal Component '
(num2str(scorenumber))]),'FontSize',18,'Rotation',22);
ylabel (strcat(['Principal Component ' (num2str(scorenumber +
1))]),'FontSize',18,'Rotation',-35);
zlabel (strcat(['Principal Component ' (num2str(scorenumber + 2))]),'FontSize',18);
title (strcat(['Clusters from PC ' (num2str(scorenumber))]),'FontSize',20);

view(3)
axis vis3d

pause(0.25);
saveas (h,strcat(['Clusters from PC ' (num2str(scorenumber)) '.bmp']));

hold off
h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
set(axes,'FontSize',14);
hold


plotstr = ['bo';'ro';'go';'co';'mo';'yo';'ko'];

for i = 1:1:(ceil(plotvaluesrows/7)),
   plotstr = [plotstr ; 'bo';'ro';'go';'co';'mo';'yo';'ko'];
end

for i = 1:1:plotvaluesrows,

plot3(means(i,scorenumber),means(i,(scorenumber+1)),means(i,(scorenumber+2)),plotstr
(i,:));
end
```

```
xlabel (strcat(['Principal Component '
(num2str(scorenumber))]),'FontSize',18,'Rotation',22);
ylabel (strcat(['Principal Component ' (num2str(scorenumber +
1))]),'FontSize',18,'Rotation',-35);
zlabel (strcat(['Principal Component ' (num2str(scorenumber + 2))]),'FontSize',18);
title (strcat(['Cluster Centers from PC ' (num2str(scorenumber))]),'FontSize',20);

view(3)
axis vis3d

pause(0.25);
saveas (h,strcat(['Cluster Centers from PC ' (num2str(scorenumber)) '.bmp']));

hold off

end
function finished

button = questdlg('Is the Quantile-Quantile Plot Appropiate? Should Image Creation
Begin?');

for i = 1:1:3,
   beep;
   pause (0.3);
end

if length(button) == 2,
   fprintf('\r Please Adjust Input Parameters Appropriately \r\r ');
   clear all;
   return;
end
if length(button) == 6,
   close all hidden;
   clear all;
   return;
end

close all hidden;   %   Closes all figures
pause (0.01);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('Calculating Loadings.\n')

[junk,dataindex] = sort(scoresindex);
```

```
clear junk
data = [dataindex,data];
[aa,bb] = size (data);
data = sortrows (data,1);
data = data(:,2:bb);
assignin('base','ordereddata',data)    %   outputs means to the workspace

for ii = 1:1:plotvaluesrows,

    tnspec = (data(plotvalues(ii,1):plotvalues(ii,2),:));
    newspec = mean(tnspec + 0.01);
    [rtnspec,rtnnew,datafval] = simcaqqplot(tnspec,loads,newspec,scorenumber,s);
    if ii == 1;
        rtnnewspec = mean(rtnnew);
    else
        rtnnewspec = [rtnnewspec;mean(rtnnew)];
    end

    h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
    set(axes,'FontSize',14);
    hold on
    plot (wavelengths,mean(rtnnew)) %mean
    title (strvcat(strcat(['Loadings Correlating to the Center']) , strcat(['of Cluster '
(num2str(plotvalues(ii,1))) ' to ' (num2str(plotvalues(ii,2))) ' Weighted by PC '
(num2str(scorenumber))])),'FontSize',20);

    xlabel('Energy (cm^-^1)','FontSize',18);
    ylabel('Loading Intensity','FontSize',18);

    pause(0.25);
    saveas (h,strcat(['Loadings Correlating to the Center of Cluster '
(num2str(plotvalues(ii,1))) ' to ' (num2str(plotvalues(ii,2))) ' weighted by PC '
(num2str(scorenumber)) '.bmp']));
    pause(0.25);

    hold off
    close(h)
    pause (0.01);
end

clear rtnnewspec
clear rtnnew
clear datafval
```

```
%%%
```

```
    end

    % assignin('base','distanceout',distanceout)

    h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
    set(axes,'FontSize',14);
    hold on
    imagesc (distanceout,clim)
    title (strvcat(strcat(['Image of the Distances from the Center']) , strcat(['of Scores '
(num2str(plotvalues(ii,1))) ' to ' (num2str(plotvalues(ii,2))) ' using PC '
(num2str(scorenumber))])),'FontSize',20);

    axis off
    pause(0.25);
    saveas (h,strcat(['Image of the Distances from the Center of Scores '
(num2str(plotvalues(ii,1))) ' to ' (num2str(plotvalues(ii,2))) ' using PC '
(num2str(scorenumber)) '.bmp']));
    pause(0.25);

    hold off
    close(h)
    pause (0.01);
end



scores = [scoresindex,scores];
[aa,bb] = size (scores);
scores = sortrows (scores,1);
scores = scores(:,2:bb);

    sorted = sortrows(scores,scorenumber);
    clim(1,1) = sorted(round(0.05*(length(scores))),scorenumber);
    clim(1,2) = sorted(round(0.95*(length(scores))),scorenumber);
    % assignin('base','sorted',sorted)
    clear sorted


i = 1;
for a = 1:1:height,
    for b = 1:1:width,
        distanceout(a,b) = scores (i,1);
        i = i + 1;
    end
end
```

```
h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
set(axes,'FontSize',14);
hold on
imagesc (distanceout,clim)
colormap(jet)
title (strcat(['Image Recreated from PC ' (num2str(scorenumber))]),'FontSize',20);

axis off
pause(0.25);
saveas (h,strcat(['Image Recreated from PC ' (num2str(scorenumber)) '.bmp']));

hold off


for i = 1:1:4,
   beep;
   pause (0.3);
end



clear all;
end
```

Function A4 - SAQQ2way

This version of SAQQ calculates best fit lines starting from the left as well as starting from the right of the QQ plot. Averaging the difference between the resulting endpoints gives a better estimate of the boundaries to the subcluster.

```
function h = SAQQ2way(data,loads,s,scores,scorenumber,optr2,percentpoints,normal)
%   function h =
SAQQ2way(data,loads,s,scores,scorenumber,optr2,percentpoints,normal)
%
%   where:
%      data is the original data set.
%
%      loads is the loadings from optPCA.
%
%      s is the s from optPCA.
%
%      scores is the array of scores from spectralPCA or any other
%            generic pca.  Each set of scores is in each column.
%
%      scorenumber is the number of the column in scores from which the
%            quantile-quantile plot is made.
%
%      optr2 is the minimum r-sqared value allowed for a line through the
%            data of the quantile-quantile plot.  This value is used to
%            determine the endpoint of each cluster.  If there are too
%            many clusters in the result, decrease rsqr.  If there are
%            too few clusters, increase rsqr.
%               *** DEFAULT optr2 = 0.9 ***
%
%     percentpoints is a value from 0 to 100 percent that represents the
%            minimum length of each line segment.  As a result, this
%            defines the minimum size of each cluster.
%               *** DEFAULT percentpoints = 10 ***
%
%      normal is a column of data representing a normal distribution for
%            plotting scores against.  If this is not specified, a
%            standard normal will be used. The number of rows in normal
%            must equal the number of rows in scores(:,scorenumber).
%            *** WARNING - discontinuities in this data set
%            will result in erroneous results. ***
%               *** DEFAULT "normal" is a standard normal distribution ***
%
%
%
%   scoresqqplot displays an empirical quantile-quantile plot and then
```

```
%   estimates clusters from the quantile-quantile plot.  To do this
%   scoresqqplot uses the r^2 value from linear regression to isolate
%   linear portions of the plot as well as checking for discontinuities in
%   the verticle direction.  The starting point for finding the first
%   cluster is defined as the left most point on the quantile-quantile
%   plot, which is also the minimum.  The first endpoint for linear
%   regression is defined as the begining point plus the number of points
%   times percentpoints or nxend = (nxbeg + (round(nx*(percentpoints/100)))).
%   The resulting r-squared value is checked against the predetermined rsqr
%   value.  If the result is larger than rsqr, then nxend is incremented by
%   the predetermined increment size.  Once resultant r-squrare drops below
%   the input rsqr, then the loop is halted and the beginning and endpoints
%   are reset.  The beginning point of the next cluster becomes the end
%   point plus one and the endpoint again becomes the begining point plus
%   the number of points times percentpoints or nxbeg = nxend + 1 and
%   nxend = (nxbeg + (round(nx*(percentpoints)))).  This continues until
%   the end point equals the number of points.
%
%
%
%   QQPLOT(scores,scorenumber) makes an empirical QQ-plot of
%   the quantiles of the data set scores(normal) versus the standard
%   normal.
%
%   QQPLOT(scores,scorenumber,normal) makes an empirical QQ-plot of
%   the quantiles of the data set scores(scorenumber1) versus the quantiles
%   of the data set normal.  This plot assumes that
%   scores normal is normal, if scores(normal) is not normal
%   then this function will likely give erroneous results.
%
%
%   The default quantiles are those of the smaller data set.
%
%
%   The purpose of the quantile-quantile plot is to determine whether
%   the sample in scorenumber1 is drawn from a Normal (i.e., Gaussian)
%   distribution, or whether the samples in scorenumber1 and normal
%   come from the same distribution type.  If the samples do come from the
%   same distribution (same shape), even if one distribution is shifted
%   and re-scaled from the other (different location and scale parameters),
%   the plot will be linear.
%
%   Heavily Edited version by Justin C. Harris from MatLab's routine...
%   Copyright 1993-2004 The MathWorks, Inc.
%   $Revision: 2.13.2.1 $  $Date: 2004/01/24 09:36:41 $
```

```
tic;
close all hidden    %   Closes all figures before the the program is run
pause (0.01);
fprintf('Calculating Quantile-Quantile Plots and QQ Plot Linear Regression.\n')

%   Checks to see if any absorbtion values are below zero and adds a
%   baseline correction if there are values below zero.

minscores = min(min(scores));   %   Finds the minimum value of scores

if minscores <= 0,  %   If the minimum value of scores is less than or equal to zero...
   scores = (scores - minscores + 0.0001); %   Add a baseline correction so the lowest
value is 0.0001.
end

if nargin <= 4 | nargin >= 9,   %   This checks to make sure that the number of inputs is
appropriate
error('Input must contain four to seven parameters.');   %   Otherwise it responds with an
error

%   If the number of inputs is 5, 6, or 7, then scores(scorenumber1) will
%   be plotted versus the standard normal.  optr2 and percentpoints values
%   will be set to their defaults if these values have not been defined.

elseif nargin >= 5 && nargin <=7,

   x = scores(:,scorenumber);  %   x is set from the input parameters to the appropriate
column of data
   y  =  sort(x);  %   y is set to the sorted values of x

%    PLOTPOS Computes plotting positions for a probability plot
%
%    PP = PLOTPOS(SX) computes the plotting positions for a probabilty
%    plot of the columns of SX (or for SX itself if it is a vector).
%    SX must be sorted before being passed into PLOTPOS.  The ith
%    value of SX has plotting position (i-0.5)/n, where n is
%    the number of rows of SX.  NaN values are removed before
%    computing the plotting positions.

   x  = plotpos(y);

%    NORMINV Inverse of the normal cumulative distribution function (cdf).
%
%    X = NORMINV(P,MU,SIGMA) returns the inverse cdf for the normal
```

```
%     distribution with mean MU and standard deviation SIGMA, evaluated at
%     the values in P.  The size of X is the common size of the input
%     arguments.  A scalar input functions as a constant matrix of the same
%     size as the other inputs.
%
%     Default values for MU and SIGMA are 0 and 1, respectively.
%
   x  = norminv(x);
   xx = x; %   Sets xx to the new values of x
   yy = y; %   Sets yy to the new values of y

   nx = size (x,1);    %   sets nx to the number of rows of x
   ny = size (y,1);    %   sets ny to the number of rows of y


%   Sets the defaults for optr2 and percentpoints if they are not given in
%   the input parameters.  This occurs for inputs of 2 or 3 parameters.  If
%   the number of parameters input is 4, then both of these are defined.
%   If the number of parameters input is 5, then it would not fall within
%   this portion of the if statement.


   if nargin == 5,     %  If the number or parameters input is two, then set optr2 and
percentpoints to their default values
       optr2 = 0.9;    %  Sets the maximum r^2 value the best fit line may have to its
default value
       percentpoints = 10; %   Sets the minimum line segment length to its default (10% of
the total data set)
   elseif nargin == 6,  %  If the number or parameters input is three, then set
percentpoints to its default value
       percentpoints = 10; %   Sets the minimum line segment length to its default (10% of
the total data set)
   end

%   Otherwise the number of input parameters is 7,  which makes an
%   empirical QQ-plot of the quantiles of the data set scores(scorenumber1)
%   versus the quantiles of the data set normal.

else
   y = scores(:,scorenumber);  %   y is set from the input parameters to the appropriate
column of data
   x = normal; %   x is set from the input parameters to the appropriate column of data

   nx = size (x,1);    %   sets nx to the number of rows of x
   ny = size (y,1);    %   sets ny to the number of rows of y
```

```
end

%   Check for errors in the input data.

if optr2 <= 0 | optr2 >= 1, %   if optr2 is not between 0 and 1 then an error occurs
   error('rsqr must be between 0 and 1');  %   displays error message
end

%   if percentpoints multiplied by the number of points is not between two
%   data points and the total number of points then an error occurs
if ceil(nx*percentpoints/100) <= 2 | ceil(nx*percentpoints/100) >= nx,
   %   displays error message
   error('percentpoints multiplied by the number of points must be between two data
points and the total number of points')
end

%   if the number of rows in the x and y arrays, corresponding to input
%   data scores and normal are not equal then an error occurs
if nx~=ny,
   %   displays error message
   error(strcat(['The number of rows in scores(:,' num2str(scorenumber) ') must equal the
number of rows in normal.']));
end

xx = sort(x);   %   sets xx to the sorted values of x
yy = sort(y);   %   sets yy to the sorted values of y


%outputs the xy values to the workspace as array qqplotdata
qqplotdata(:,1) = (xx); %   sets column 1 of qqplotdata to the xx values
qqplotdata(:,2) = (yy); %   sets column 2 of qqplotdata to the yy values
assignin('base','qqplotdata',qqplotdata)    %   outputs qqplotdata to the workspace
clear qqplotdata


q1x = prctile(x,25);
q3x = prctile(x,75);
q1y = prctile(y,25);
q3y = prctile(y,75);
qx = [q1x; q3x];
qy = [q1y; q3y];


dx = q3x - q1x;
```

```
dy = q3y - q1y;
slope = dy./dx;
centerx = (q1x + q3x)/2;
centery = (q1y + q3y)/2;
maxx = max(x);
minx = min(x);
maxy = centery + slope.*(maxx - centerx);
miny = centery - slope.*(centerx - minx);

mx = [minx; maxx];
my = [miny; maxy];



    decnxend = 1;    %  sets the amount nxend is increased each iteration
   minend = percentpoints/100; %  multiplies by number of scores for setting the
minimum best fit line
   nxbeg = 1;  %   set nxbeg to the beginning of the array
   nxend = 1;  %   set nxend to the beginning of the array
   plotvalues = 0; %   initiates a new variable
   r2 = 1; %   sets the r2 values for the initial while test

while nxend ~= nx + 1,

   if nxend < (nx - (ceil(nx*minend)));
      nxend = nxbeg + (ceil(nx*minend));
   else
      nxend = nx;
   end

   i = nxbeg;

    while  r2 > optr2 && nxend ~= nx + 1,   %   until r^2 reaches this minimum value,
loop

      if (yy(nxend)-yy(nxend-1)) > (yy(nxend-1)-yy(nxbeg)) , % or % if (yy(nxend)-
yy(nxend-1)) > (yy(nxend-1)-yy(ceil(0.5*nx*minend))) ,
         if plotvalues == 0,
            plotvalues = [nxbeg,(nxend - 1)];    %   create array and output the beginning
and ending values to an array
            nxend = nxend + 1;
         elseif nxend >= nx,
            if nxend == nx,
               plotvalues = [plotvalues;[(nxbeg-1),(nxend)]];
               nxend = nxend + 1;
```

```
            else
               plotvalues = [plotvalues;[(nxbeg-1),(nx)]];
               nxend = nx + 1;
            end
         else
            plotvalues = [plotvalues;[(nxbeg-1),(nxend-1)]];   %   otherwise, append
beginning and ending values to the array
            nxend = nxend + 1;
         end
         nxbeg = nxend - 1;
         r2 = 0;
      else,
         xx2 = (xx(nxbeg:nxend));  %   sets the x value for input into (getrsqr)
         yy2 = (yy(nxbeg:nxend));  %   sets the y value for input into (getrsqr)

         r2 = getrsqr(xx2,yy2);  %   calculates bestfit and r^2

         if   r2 > optr2 && nxend ~= nx,  %   if r^2 is less than the minimum r^2
            if nx >= nxend + decnxend,
               nxend = nxend + decnxend; %   shortens the number of points for fitting next
iteration
            else
               nxend = nx;
            end

         elseif plotvalues == 0,
            if r2 <= optr2,
               plotvalues = [nxbeg,(nxend-1)];    %   create array and output the beginning
and ending values to an array
               nxend = nxend + 1;
            else,
               plotvalues = [nxbeg,nxend];
               nxend = nxend + 1;
            end

         elseif nxend == nx,
            plotvalues = [plotvalues;[(nxbeg-1),(nxend)]];
            nxend = nxend + 1;
         else
            plotvalues = [plotvalues;[(nxbeg-1),(nxend-1)]];   %   otherwise, append
beginning and ending values to the array
            nxend = nxend + 1;
         end
      end
   end
```

```
    if nxend ~= nx,
       nxbeg = nxend;  %   nxbeginning is set equal to nxend
       r2 = 1;
    end

    time = (round(toc*10))/10;
    fprintf('Elapsed time is % 5.1f seconds.\n', time)

end



plotvalues1 = plotvalues


    decnxend = 1;    %  sets the amount nxend is increased each iteration
    minend = percentpoints/100; %  multiplies by number of scores for setting the
minimum best fit line
    nxbeg = nx;  %   set nxbeg to the end of the array
    nxend = nx;  %   set nxend to the end of the array
    plotvalues = 0; %   initiates a new variable
    r2 = 1; %   sets the r2 values for the initial while test

while nxend ~= 0,

    if nxend > (0 + (ceil(nx*minend)));
       nxend = nxbeg - (ceil(nx*minend));
    else
       nxend = 1;
    end

    i = nxbeg;

     while  r2 > optr2 && nxend ~= 0,   %   until r^2 reaches this minimum value, loop

       if (yy(nxend+1)-yy(nxend)) > (yy(nxbeg)-yy(nxend+1)) , % or % if (yy(nxend)-
yy(nxend-1)) > (yy(nxend-1)-yy(ceil(0.5*nx*minend))) ,
          if plotvalues == 0,
             plotvalues = [(nxend + 1),nxbeg];    %   create array and output the beginning
and ending values to an array
             %nxend = nxend - 1;
          elseif nxend <= 1,
             if nxend == 1,
                plotvalues = [plotvalues;[(nxend),(nxbeg-1)]];
```

```
            nxend = nxend - 1;
          else
            plotvalues = [plotvalues;[1,(nxbeg-1)]];
            nxend = 0;
          end
        else
          plotvalues = [plotvalues;[(nxend+1),(nxbeg)]];   %   otherwise, append
beginning and ending values to the array
          %nxend = nxend - 1;
        end
        nxbeg = nxend - 1;
        r2 = 0;
      else,
        xx2 = (xx(nxend:nxbeg));  %   sets the x value for input into (getrsqr)
        yy2 = (yy(nxend:nxbeg));  %   sets the y value for input into (getrsqr)

        r2 = getrsqr(xx2,yy2);  %   calculates bestfit and r^2

        if   r2 > optr2 && nxend ~= 1,  %   if r^2 is less than the minimum r^2
          if 0 <= nxend - decnxend,
            nxend = nxend - decnxend; %   shortens the number of points for fitting next
iteration
          else
            nxend = 1;
          end

        elseif plotvalues == 0,
          if r2 <= optr2,
            plotvalues = [(nxend),nxbeg];    %   create array and output the beginning
and ending values to an array
            nxend = nxend - 1;
          else,
            plotvalues = [nxend,nxbeg];
            nxend = nxend - 1;
          end

        elseif nxend == 1,
          plotvalues = [plotvalues;[(nxend),(nxbeg)]];
          nxend = nxend - 1;
        else
          plotvalues = [plotvalues;[(nxend),(nxbeg)]];   %   otherwise, append beginning
and ending values to the array
          nxend = nxend - 1;
        end
      end
```

```
    end

    if nxend ~= 0,
       nxbeg = nxend;  %   nxbeginning is set equal to nxend
       r2 = 1;
    end

    time = (round(toc*10))/10;
    fprintf('Elapsed time is % 5.1f seconds.\n', time)

end

plotvalues2 = rot90(plotvalues',1);

plotvalues2



plotvalues = round((plotvalues1+plotvalues2).*(1/2));

plotvalues



scrsz = get(0,'ScreenSize');
h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
set(axes,'FontSize',14);

hold

plotvaluesrows = size(plotvalues,1);

plotstr = ['b.';'r.';'g.';'c.';'m.';'y.';'k.'];

for i = 1:1:(ceil(plotvaluesrows/7)),
    plotstr = [plotstr ; 'b.';'r.';'g.';'c.';'m.';'y.';'k.'];
end

for i = 1:1:plotvaluesrows,

plot(xx(plotvalues(i,1):plotvalues(i,2)),yy(plotvalues(i,1):plotvalues(i,2)),plotstr(i,:),qx,q
y,'-',mx,my,'-.');
end
```

```
xlabel('X Quantiles','FontSize',18);
ylabel('Y Quantiles','FontSize',18);


title (strcat(['QQ Plot from PC ' (num2str(scorenumber))]),'FontSize',20);

saveas (h,strcat(['QQ Plot from PC ' (num2str(scorenumber)) '.bmp']))

hold off

h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
set(axes,'FontSize',14);
hold

for i = 1:1:plotvaluesrows,

plot(xx(plotvalues(i,1):plotvalues(i,2)),yy(plotvalues(i,1):plotvalues(i,2)),plotstr(i,:),qx,q
y,'-',mx,my,'-.');
   getresid(xx(plotvalues(i,1):plotvalues(i,2)),yy(plotvalues(i,1):plotvalues(i,2)));
end

xlabel('X Quantiles','FontSize',18);
ylabel('Y Quantiles','FontSize',18);
title (strcat(['QQ Plot with Best Fit Lines and Residuals from PC '
(num2str(scorenumber))]),'FontSize',20);

saveas (h,strcat(['QQ Plot with Best Fit Lines and Residuals from PC '
(num2str(scorenumber)) '.bmp']));

hold off

xx = xx2;
yy = yy2;




[scores,scoresindex] = sortrows(scores,scorenumber); % sorts the x score rows to
correlate with the qqplot x output

h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
set(axes,'FontSize',14);
hold
```

```
for i = 1:1:plotvaluesrows,

plot3(scores(plotvalues(i,1):plotvalues(i,2),scorenumber),scores(plotvalues(i,1):plotvalue
s(i,2),(scorenumber+1)),scores(plotvalues(i,1):plotvalues(i,2),(scorenumber+2)),plotstr(i,:
));
   if i == 1,
      means = mean(scores(plotvalues(i,1):plotvalues(i,2),:));
   else,
      means = [means;(mean(scores(plotvalues(i,1):plotvalues(i,2),:)))];
   end
end
assignin('base','means',means)    %   outputs means to the workspace


xlabel (strcat(['Principal Component '
(num2str(scorenumber))]),'FontSize',18,'Rotation',22);
ylabel (strcat(['Principal Component ' (num2str(scorenumber +
1))]),'FontSize',18,'Rotation',-35);
zlabel (strcat(['Principal Component ' (num2str(scorenumber + 2))]),'FontSize',18);
title (strcat(['Clusters from PC ' (num2str(scorenumber))]),'FontSize',20);

view(3)
axis vis3d

pause(0.25);
saveas (h,strcat(['Clusters from PC ' (num2str(scorenumber)) '.bmp']));

hold off
h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
set(axes,'FontSize',14);
hold


plotstr = ['bo';'ro';'go';'co';'mo';'yo';'ko'];

for i = 1:1:(ceil(plotvaluesrows/7)),
   plotstr = [plotstr ; 'bo';'ro';'go';'co';'mo';'yo';'ko'];
end

for i = 1:1:plotvaluesrows,

plot3(means(i,scorenumber),means(i,(scorenumber+1)),means(i,(scorenumber+2)),plotstr
(i,:));
end
```

```
xlabel (strcat(['Principal Component '
(num2str(scorenumber))]),'FontSize',18,'Rotation',22);
ylabel (strcat(['Principal Component ' (num2str(scorenumber +
1))]),'FontSize',18,'Rotation',-35);
zlabel (strcat(['Principal Component ' (num2str(scorenumber + 2))]),'FontSize',18);
title (strcat(['Cluster Centers from PC ' (num2str(scorenumber))]),'FontSize',20);

view(3)
axis vis3d

pause(0.25);
saveas (h,strcat(['Cluster Centers from PC ' (num2str(scorenumber)) '.bmp']));

hold off

for i = 1:1:3,
   beep;
   pause (0.3);
end

button = questdlg('Is the Quantile-Quantile Plot Appropiate? Should Image Creation
Begin?');

if length(button) == 2,
   fprintf('\r Please Adjust Input Parameters Appropriately \r\r ');
   clear all;
   return;
end
if length(button) == 6,
   close all hidden;
   clear all;
   return;
end

close all hidden;   %   Closes all figures
pause (0.01);
%%%
```

```
data = data(:,2:bb);
assignin('base','ordereddata',data)    %   outputs means to the workspace

for ii = 1:1:plotvaluesrows,

   tnspec = (data(plotvalues(ii,1):plotvalues(ii,2),:));
   newspec = mean(tnspec + 0.01);
   [rtnspec,rtnnew,datafval] = simcaqqplot(tnspec,loads,newspec,scorenumber,s);
   if ii == 1;
      rtnnewspec = mean(rtnnew);
   else
      rtnnewspec = [rtnnewspec;mean(rtnnew)];
   end

   h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
   set(axes,'FontSize',14);
   hold on
   plot ((1100:10:1700),mean(rtnnew)) %mean
   title (strvcat(strcat(['Loadings Correlating to the Center']) , strcat(['of Cluster '
(num2str(plotvalues(ii,1))) ' to ' (num2str(plotvalues(ii,2))) ' Weighted by PC '
(num2str(scorenumber))])),'FontSize',20);

   xlabel('Energy (cm^-^1)','FontSize',18);
   ylabel('Loading Intensity','FontSize',18);

   pause(0.25);
   saveas (h,strcat(['Loadings Correlating to the Center of Cluster '
(num2str(plotvalues(ii,1))) ' to ' (num2str(plotvalues(ii,2))) ' weighted by PC '
(num2str(scorenumber)) '.bmp']));
   pause(0.25);

   hold off
   close(h)
   pause (0.01);
end

clear rtnnewspec
clear rtnnew
clear datafval

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

width = 20;  %101; %103
height = 43;  %127; %96
```

```
for ii = 1:1:plotvaluesrows,
   b = nx;
   tnspec = scores(plotvalues(ii,1):plotvalues(ii,2),:);    %
scores(plotvalues(1,1):plotvalues(1,2),:);
   radfrac = 0.4;
   sensitiv = 0.001;


   [btrain,cnter]=replica(tnspec,b);

   for i = 1:1:nx,
      newspec = scores (i,:);
      [sds,sdskew] = qb(tnspec,btrain,newspec,cnter,radfrac,sensitiv);
      if i == 1;
         distance = sds;
      else
         distance = [distance;sds];
      end
   end

   time = (round(toc/6))/10;
   fprintf('Elapsed time is % 5.1f minutes.\n', time)


   distance = [scoresindex,distance];
   [aa,bb] = size (distance);
   distance = sortrows (distance,1);
   distance = distance(:,2:bb);



   % assignin('base','distance',distance)    %   outputs means to the workspace



   i = 1;
   for a = 1:1:height,  %   96
      for b = 1:1:width,  %   103
         distanceout(a,b) = distance (i,1);
         i = i + 1;
      end
   end
```

```
    h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
    set(axes,'FontSize',14);
    hold on
    imagesc (distanceout)
    title (strvcat(strcat(['Image of the Distances from the Center']) , strcat(['of Scores '
(num2str(plotvalues(ii,1))) ' to ' (num2str(plotvalues(ii,2))) ' using PC '
(num2str(scorenumber))])),'FontSize',20);

    axis off
    pause(0.25);
    saveas (h,strcat(['Image of the Distances from the Center of Scores '
(num2str(plotvalues(ii,1))) ' to ' (num2str(plotvalues(ii,2))) ' using PC '
(num2str(scorenumber)) '.bmp']));
    pause(0.25);

    hold off
    close(h)
    pause (0.01);
end



scores = [scoresindex,scores];
[aa,bb] = size (scores);
scores = sortrows (scores,1);
scores = scores(:,2:bb);

i = 1;
for a = 1:1:height,
    for b = 1:1:width,
        distanceout(a,b) = scores (i,1);
        i = i + 1;
    end
end

h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
set(axes,'FontSize',14);
hold on
imagesc (distanceout)
colormap(jet)
title (strcat(['Image Recreated from PC ' (num2str(scorenumber))]),'FontSize',20);

axis off
pause(0.25);
saveas (h,strcat(['Image Recreated from PC ' (num2str(scorenumber)) '.bmp']));
```

```
hold off


for i = 1:1:4,
   beep;
   pause (0.3);
end



clear all;
end
```

Function A5 - optPCA

```
function optPCA(data,lv)

%PCA Principal components analysis optimized
%  PCA uses svd to perform pca on a data matrix. It is
%  assumed that samples are rows and variables are columns.
%  The input is the data matrix (data). Outputs are the scores
%  (scores) and loadings (loads).
tic
a = 1;
[m,n] = size(data);

 if (lv > min([m n])) %& (plots ~= 0)
   disp('Resetting lv to be equal to min([m n])')
   lv = min([m n]);
 end
toc
if n < m
   cov = (data'*data)/(m-1);
   [u,s,v] = svdc(cov);
   assignin('base','u',u)  %   outputs u to the workspace
   assignin('base','s',s)  %   outputs s to the workspace
   assignin('base','v',v)  %   outputs v to the workspace
   u = 0;
   s = 0;
   cov = 0;

   toc


   %  Form the PCA Model Based on the Number of PCs Chosen
   loads  = v(:,1:lv);
   toc
   scores = data*loads;
   toc
else,
   [scores,loads,ssq,res,reslm,tsqlm,tsq] = pca(data,0,0,lv);
end
loads = loads';
assignin('base','scores',scores)    %   outputs scores to the workspace
assignin('base','loads',loads)    %   outputs loads to the workspace
```

```
figure
hold
plot3 (scores(:,1),scores(:,2),scores(:,3),'b.');

xlabel('PC1');
ylabel('PC2');
zlabel('PC3');
```

Function A6 - spectralPCA

```
function spectralPCA(spectraldata)
%spectraldata = load(file);
%spectraldatamc = spectraldata - repmat(mean(spectraldata),9888,1);
spectraldatamc = zscore(spectraldata);
[scores,loads,ssq,res,reslm,tsqlm,tsq] = pca(spectraldatamc,0,0,12);
plot3 (scores(:,1),scores(:,2),scores(:,3),'k.');

xlabel('PC1');
ylabel('PC2');
zlabel('PC3');

ssqtable(ssq, 12);

hist (scores(:,1:10))


%R = zeros(24);
% for i = 1:6
%     R(((i-1)*100+1):i*100) = i;
% end

%R = [1,2,3,1,2,3];


%pcaVar = [94.18,2.38, 1.90];
%pcellipse(scores, R, 2, pcaVar, [1,2,3]);

% data = principal component scores
% R = concentration or group vector
% nosd = number of standard deviations to include
% s = % of the variance from pca
% take = pcs to include in the model, for largest separation use take =
% [1,2,3]
```

Function A7 - getresid

```
function [dis, slope, intercept] = getresid(x,y);
%A Function which gets the r-square (r^2) between array and its best-fit
%(least squares) line as well as calculated the residuals and plots data
%and the fit and the residuals
%y is an n-by-1 vector of observations
%x is an n-by-p matrix of regressors
%getrsqr compares the y values to obtain an r^2 value

ny = length(y);                    %gets the number of rows in column y
bestfit = polyfit (x,y,1);          %gets the bestfit polynomial values for x an y

a = bestfit(1,1);  % sets the slope
b = -1;
c = bestfit(1,2);  % sets the intercept
refy = a*x + c;
rows = size(x);  %counts the number of rows in the array
for I = 1:1:rows,

   p = [(x(I,1)),(y(I,1))];
   dist = line_imp_point_dist_2d(a,b,c,p);
   dis(I,1) = dist;

end

plot(x,dis(:,1),'g', x,refy,'r')

slope = a;
intercept = c;
```

Function A8 - getrsqr

```
function [rsqr, slope, intercept] = getrsqr(x, y);
%A Function which gets the r-square (r^2) between matrix and its best-fit
%(least squares) line
%y is an n-by-1 vector of observations
%x is an n-by-p matrix of regressors
%getrsqr compares the y values to obtain an r^2 value

ny = length(y);                    %gets the number of rows in column y
bestfit = polyfit (x,y,1);          %gets the bestfit polynomial values for x an y
yp = polyval(bestfit,x);            %gets the bestfit polynomial values between the
bestfit line and x
yp = [ones(ny,1),yp];               %places a column of ones in front of yp, and moves
yp to the second column
[b,bint,r,rint,stats] = regress(y,yp);     %regresses y and yp (see function regress)
rsqr = stats(1);                   %returns the rsqr value from regress
slope = bestfit(1,1);               % sets the slope
intercept = bestfit(1,2);           % sets the intercept
```

Function A9 - convertdata2d

```
function h = convertdata2d(data, width, height)
[x,y]=size(data);
%data = flipud(data);
data2d = zeros(height,width,y);
i = 1;

for a = 1:1:width,
   for b = 1:1:height,
      data2d(b,a,:) = data(i,:);
      i = i + 1;
   end
end
data2d = flipdim(data2d,1);
assignin('base','data2d',data2d);

imagedata2d = sum(data2d,3);

assignin('base','imagedata2d',imagedata2d);

sorted = zeros(height*width,1);

i = 1;


for a = 1:1:height,
   for b = 1:1:width,
      sorted(i,1) = imagedata2d(a,b);
      i = i + 1;
   end
end

sorted = sort(sorted);
clim(1,1) = sorted(round(0.05*(length(sorted))),1);
clim(1,2) = sorted(round(0.95*(length(sorted))),1);

clear sorted
imagesc(imagedata2d);
figure
imagesc(imagedata2d,clim);

end
```

Function A10 - data2dtoimage

```
function h = data2dtoimage(data2d)

close all hidden

imagedata2d = sum(data2d,3);

assignin('base','imagedata2d',imagedata2d);

figure
imagesc(imagedata2d);

[height,width,z] = size(data2d);

sorted = zeros(height*width,1);

i = 1;


for a = 1:1:height,
   for b = 1:1:width,
      sorted(i,1) = imagedata2d(a,b);
      i = i + 1;
   end
end

sorted = sort(sorted);
clim(1,1) = sorted(round(0.05*(length(sorted))),1);
clim(1,2) = sorted(round(0.95*(length(sorted))),1);

%imagedata2d(25,25) = 999999;

clear sorted
figure
imagesc(imagedata2d,clim);

end
```

Function A11 - polyline

```
%Made by Clay Harris and Rob Dolan June 9, 2005 at way early in the morning
%Calculates the shortest vector from a point within a matrix to a line
%m is the slop of the line and b is the y-intercept.
%datamatrix is the matrix which contains the x and y coordinates
%X coordinates must be in column 1, and Y coordinates must be in column 2


function polyfitlinedistance = polyline(datamatrix)
%function distancetoaline = lindistance(a, b, datamatrix)

x = datamatrix(:,1);
y = datamatrix(:,2);

bestfit = polyfit(x,y,1);

b = bestfit(1,1);
a = bestfit(1,2);



rows = size(datamatrix,1);
for I = 1:1:rows,

   d  = abs((((x(I,1))*a)+(y(I,1))+b)/(sqrt((a^2)+(1))));
   distance(I,1) = d;

end

assignin('base','distance',distance);
plot(x,distance(:,1),'g')
hold
plot(x,y,'r')
refline(a,b)
```

Function A12 - plotpos

```
function [pp,n] = plotpos(sx)
%PLOTPOS Compute plotting positions for a probability plot
%   PP = PLOTPOS(SX) compute the plotting positions for a probabilty
%   plot of the columns of SX (or for SX itself if it is a vector).
%   SX must be sorted before being passed into PLOTPOS.  The ith
%   value of SX has plotting position (i-0.5)/n, where n is
%   the number of rows of SX.  NaN values are removed before
%   computing the plotting positions.
%
%   [PP,N] = PLOTPOS(SX) also returns N, the largest sample size
%   among the columns of SX.  This N can be used to set axis limits.

[n, m] = size(sx);
if n == 1
  sx = sx';
  n = m;
  m = 1;
end

nvec = sum(~isnan(sx));
pp = repmat((1:n)', 1, m);
pp = (pp-.5) ./ repmat(nvec, n, 1);
pp(isnan(sx)) = NaN;

if (nargout > 1)
  n = max(nvec);  % sample size for setting axis limits
end
```

Function A13 – projection

```
function projection(scores,a,b,c)

% where a, b, and c are the score numbers

close all hidden
cluster1mean = mean (scores(1:100,:));
cluster2mean = mean (scores(101:200,:));

meanx1 = cluster1mean(1,a);
meanx2 = cluster2mean(1,a);
meany1 = cluster1mean(1,b);
meany2 = cluster2mean(1,b);
meanz1 = cluster1mean(1,c);
meanz2 = cluster2mean(1,c);

cluster1std = (std (scores(1:100,:))) .* 2;
cluster2std = (std (scores(101:200,:))) .* 2;

std1x = cluster1std(1,a);
std2x = cluster2std(1,a);
std1y = cluster1std(1,b);
std2y = cluster2std(1,b);
std1z = cluster1std(1,c);
std2z = cluster2std(1,c);

[xx,yy,zz] = sphere(1000);
zero = zeros(1,1001);

figure;
hold
view(3)
axis vis3d

xlabel ('x','FontSize',24);
ylabel ('y','FontSize',24);
zlabel ('z','FontSize',24);
% title ('Cluster Projections','FontSize',20);

x = xx(500,:)' .* std1x + meanx1;
y = yy(500,:)' .* std1y + meany1;
plot3(x,y,zero,'b','LineWidth',2); % x-y plane

y = xx(500,:)' .* std1y + meany1;
```

```
z = yy(500,:)' .* std1z + meanz1;
plot3(zero,y,z,'b','LineWidth',2); % y-z plane

x = xx(500,:)' .* std1x + meanx1;
z = yy(500,:)' .* std1z + meanz1;
plot3(x,zero,z,'b','LineWidth',2); % x-z plane

plot3(scores(1:100,a),scores(1:100,b),scores(1:100,c),'b.','LineWidth',5,'MarkerSize',16);

x = xx(500,:)' .* std2x + meanx2;
y = yy(500,:)' .* std2y + meany2;
plot3(x,y,zero,'r','LineWidth',2); % x-y plane

y = xx(500,:)' .* std2y + meany2;
z = yy(500,:)' .* std2z + meanz2;
plot3(zero,y,z,'r','LineWidth',2); % y-z plane

x = xx(500,:)' .* std2x + meanx2;
z = yy(500,:)' .* std2z + meanz2;
plot3(x,zero,z,'r','LineWidth',2); % x-z plane

plot3(scores(101:200,a),scores(101:200,b),scores(101:200,c),'r.','LineWidth',5,'MarkerSiz
e',16');


set(gca,'LineWidth',2)

set(gca,'XTick',[])
set(gca,'YTick',[])
set(gca,'ZTick',[])



end
```

Solid State Spectral Imager Functions

Function A14 – SSSIgui

```
% ------------------------------------------------------------------
function varargout = SSSIgui(varargin)
% SSSIGUI M-file for SSSIgui.fig
%      SSSIGUI, by itself, creates a new SSSIGUI or raises the existing
%      singleton*.
%
%      H = SSSIGUI returns the handle to a new SSSIGUI or the handle to
%      the existing singleton*.
%
%      SSSIGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in SSSIGUI.M with the given input arguments.
%
%      SSSIGUI('Property','Value',...) creates a new SSSIGUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before SSSIgui_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to SSSIgui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help SSSIgui

% Last Modified by GUIDE v2.5 27-Oct-2004 15:23:16
% ------------------------------------------------------------------
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @SSSIgui_OpeningFcn, ...
                   'gui_OutputFcn',  @SSSIgui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --------------------------------------------------------------------
% --- Executes just before SSSIgui is made visible.
function SSSIgui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to SSSIgui (see VARARGIN)
% --------------------------------------------------------------------
% Choose default command line output for SSSIgui
handles.output = hObject;

% Parse initialization file and setup GUI
% Initialization file 'SSSIgui.ini' must be in same directory as fig and m
% files
fid = fopen('SSSIgui.ini','rt');
if(fid == -1)
   error(['ERROR - Could not open GUI initialization file ','SSSIgui.ini']);
end
try
   %Get baud rate for serial communications
   nextline = fgetl(fid);
   if(findstr(nextline,'BAUDRATE') ~= 1)
      error('ERROR - Improper format in initialization file');
   end
   loc = findstr(nextline,'=');
   handles.search_baud = str2num(nextline(loc+1:length(nextline)));
   %Get number of leds
   nextline = fgetl(fid);
   if(findstr(nextline,'HADAMARDLEDS') ~= 1)
      error('ERROR - Improper format in initialization file');
   end
   loc = findstr(nextline,'=');
   handles.num_leds = str2num(nextline(loc+1:length(nextline)));
   %Get number of hadamard sequences
   nextline = fgetl(fid);
   if(findstr(nextline,'HADAMARDSEQS') ~= 1)
```

```
      error('ERROR - Improper format in initialization file');
   end
   loc = findstr(nextline,'=');
   handles.num_seqs = str2num(nextline(loc+1:length(nextline)));
   %Get saturation value for detector system
   nextline = fgetl(fid);
   if(findstr(nextline,'SATURATION') ~= 1)
      error('ERROR - Improper format in initialization file');
   end
   loc = findstr(nextline,'=');
   handles.saturation_value = str2num(nextline(loc+1:length(nextline)));
   %Get x-axis values corresponding to each led
   nextline = fgetl(fid);
   if(findstr(nextline,'XVALS') ~= 1)
      error('ERROR - Improper format in initialization file');
   end
   loc = findstr(nextline,'=');
   handles.xvals = str2num(nextline(loc+1:length(nextline)));
   if(length(handles.xvals) ~= handles.num_leds)
      error('ERROR - Length of XVALS does not match number of LEDs');
   end
   %Get hadamard sequences
   nextline = fgetl(fid);
   if(findstr(nextline,'SEQS') ~= 1)
      error('ERROR - Improper format in initialization file');
   end
   handles.seqs = zeros(handles.num_seqs,handles.num_leds);
   for i=1:handles.num_seqs
      handles.seqs(i,:) = str2num(fgetl(fid));
   end
catch
   fclose(fid);
   error(lasterr,'modal');
end

% Find pseudo-inverse of hadamard transform
handles.hinv = pinv(handles.seqs);

% Initializes flags in handles structure
handles.search_open = 0;
handles.saving_data = 0;
handles.scans_saved = 1;

% Create bar graph handles
axes(handles.axes1);
```

```
handles.bar_handles = zeros(5,1);
handles.bar_handles(1) = bar(handles.xvals(1:5),ones(5,1));
set(handles.bar_handles(1), 'FaceColor',[0 0 1]) %blue
hold;
handles.bar_handles(2) = bar(handles.xvals(6:10),ones(5,1));
set(handles.bar_handles(2), 'FaceColor', [0 1 0]) %green
handles.bar_handles(3) = bar(handles.xvals(11:15),ones(5,1));
set(handles.bar_handles(3), 'FaceColor', [1 1 0]) %yellow
handles.bar_handles(4) = bar(handles.xvals(16:20),ones(5,1));
set(handles.bar_handles(4), 'FaceColor', [1 0.5 0]) %orange
handles.bar_handles(5) = bar(handles.xvals(21:25),ones(5,1));
set(handles.bar_handles(5), 'FaceColor', [1 0 0]) %red

set(gca,'xtick',[1,2,3,4,5]);

% Set GUI control defaults
reset_gui(hObject,eventdata,handles);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes SSSIgui wait for user response (see UIRESUME)
% uiwait(handles.figure1);



% --------------------------------------------------------------------
% --- Outputs from this function are returned to the command line.
function varargout = SSSIgui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
% Get default command line output from handles structure
varargout{1} = handles.output;

% --------------------------------------------------------------------
% --- Executes on button press in singlebut.
function singlebut_Callback(hObject, eventdata, handles)
% hObject    handle to singlebut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
```

```
collect_single(hObject, eventdata, handles);

% --------------------------------------------------------------------
% --- Executes on button press in continuousbut.
function continuousbut_Callback(hObject, eventdata, handles)
% hObject    handle to continuousbut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
%global mvArrayLen; %Comment out if no XY stage
%global posXY; %Comment out if no XY stage
for i=1:100  % Change the i value in collect_single to match the maximum number of
scans
   if length(get(handles.continuousbut,'Enable')) == 2,
      handles = collect_single(hObject, eventdata, handles);
   else,
      gui_mainfcn(gui_State, varargin{:});
   end
end

% --------------------------------------------------------------------
% --- Collects a single spectrum from the SEARCH instrument
function handles = collect_single(hObject, eventdata, handles)
% --------------------------------------------------------------------
%moveNext(); %Comment out if no XY stage
try
   numscans = 1;
   data = zeros(handles.num_seqs,numscans);
   vals = zeros(handles.num_seqs);

   for n = 1:numscans
      fprintf(handles.search_id,'');
      commtest = 0;
      while(handles.search_id.BytesAvailable < handles.num_seqs*5)
         pause(0.01);
         commtest = commtest + 1;
         if commtest == 500,
            reset_gui(hObject, eventdata, handles)
            errordlg('Communication Error - Please verify that SSSI is connected properly
and turned on.','Communication Error');
            return
            %  figure1_CloseRequestFcn(hObject, eventdata, handles)
         end
      end
      for i=1:handles.num_seqs
```

```
            temp = fread(handles.search_id,1,'uint8');
            if(temp ~= i)
                error('1','ERROR - Invalid sequence order output from SSSI.');
            end
            temp = fread(handles.search_id,1,'uint32');
            data(i,n) = double(temp);
        end
    end

    % Average
    for i = 1:handles.num_seqs
        for n = 1:numscans
            vals(i) = vals(i) + data(i,n);
        end
        vals(i) = vals(i)/numscans;
    end
    if(handles.search_id.BytesAvailable > 0)
        dummy = fread(handles.search_id,handles.search_id.BytesAvailable);
    end
catch
    %errordlg(lasterr,'modal');
    return;
end

if(~isempty(find(vals >= handles.saturation_value)))
    errordlg('ERROR - Signal saturation encountered','modal');
    return;
end

if (handles.saving_data)
    handles.scans_saved = handles.scans_saved + 1;
end

vals = handles.hinv*vals;
for i = 1:handles.num_leds
    if vals(i,1) < 0
        vals(i,1) = 0;
    end
    handles.yvals(handles.scans_saved, i) = vals(i,1);
end

axes(handles.axes1);

for i=1:5
    set(handles.bar_handles(i),'YData',vals(((i-1)*5+1):(i*5)));
```

```
end
set(handles.scansedit,'String',num2str(handles.scans_saved));
guidata(hObject,handles);
%checkReset(); %Comment out if no XY stage


% --------------------------------------------------------------------
% --- Executes on button press in openbut.
function openbut_Callback(hObject, eventdata, handles)
% hObject    handle to openbut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
% Reset GUI
reset_gui(hObject,eventdata,handles);
% Get specified serial port from popup menu
port_contents = get(handles.portpop,'String');
handles.search_port = port_contents{get(handles.portpop,'Value')};
% Create serial object and open communications
handles.search_id =
serial(handles.search_port,'BaudRate',handles.search_baud,'Timeout',10);
try
   fopen(handles.search_id);
catch
   errordlg(['ERROR - Could not open communications on ',handles.search_port, '.
Matlab returned error: ',lasterr],'modal');
   delete(handles.search_id);
   return;
end

handles.search_open = 1;

% Test communications to see if SSSI is connected

numscans = 1;
data = zeros(handles.num_seqs,numscans);
vals = zeros(handles.num_seqs);

for n = 1:numscans
   fprintf(handles.search_id,'');
   commtest = 0;
   while(handles.search_id.BytesAvailable < handles.num_seqs*5)
      pause(0.01);
      commtest = commtest + 1;
      if commtest == 500,
```

```
            reset_gui(hObject, eventdata, handles)
            errordlg('Communication Error - Please verify that SSSI is connected properly
and turned on.','Communication Error');
            return
            %  figure1_CloseRequestFcn(hObject, eventdata, handles)
        end
    end
    for i=1:handles.num_seqs
        temp = fread(handles.search_id,1,'uint8');
        if(temp ~= i)
            error('1','ERROR - Invalid sequence order output from SSSI.');
        end
        temp = fread(handles.search_id,1,'uint32');
        data(i,n) = double(temp);
    end
end

% Average
for i = 1:handles.num_seqs
    for n = 1:numscans
        vals(i) = vals(i) + data(i,n);
    end
    vals(i) = vals(i)/numscans;
end
if(handles.search_id.BytesAvailable > 0)
    dummy = fread(handles.search_id,handles.search_id.BytesAvailable);
end



    %Comment out if no XY stage
    % Initialize XY-Stage
    %InitXY('COM4');
    %global MoveArray;
    %global posXY;
    %global mvArrayLen;
    %posXY = 0;
    %MoveArray = load('MoveArray.txt');
    %[mvArrayLen, tmp] = size(MoveArray);
    %clear tmp;
    %handles.yvals = zeros(mvArrayLen, handles.num_seqs);

    % Enable necessary GUI controls
    set(handles.singlebut,'Enable','on');
    set(handles.continuousbut,'Enable','on');
```

```
    set(handles.savefilebut,'Enable','on');
    set(handles.closebut,'Enable','on');
    set(handles.statustxt,'String','Open');
    set(handles.statustxt,'ForegroundColor',[0 1 0]);
    % Update handles structure
    guidata(hObject,handles);



% --------------------------------------------------------------------
% --- Executes on selection change in portpop.
function portpop_Callback(hObject, eventdata, handles)
% hObject    handle to portpop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns portpop contents as cell array
%        contents{get(hObject,'Value')} returns selected item from portpop
% --------------------------------------------------------------------



% --------------------------------------------------------------------
% --- Executes during object creation, after setting all properties.
function portpop_CreateFcn(hObject, eventdata, handles)
% hObject    handle to portpop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
% --------------------------------------------------------------------
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --------------------------------------------------------------------
% --- Reset GUI variables and controls
function reset_gui(hObject, eventdata, handles)
% --------------------------------------------------------------------
% Check if search is open and close
if(handles.search_open)
    fclose(handles.search_id);
```

```
    delete(handles.search_id);
    %HomeXY(); %Comment out if no XY stage
    %DisconnectXY(); %Comment out if no XY stage
end
if(handles.saving_data)
    fclose(handles.savefile_id);
    delete(handles.savefile_id);
end
handles.search_id = [];
handles.savefile_id = [];
handles.search_open = 0;
handles.saving_data = 0;
handles.scans_saved = 1;
% Set GUI control defaults
set(handles.singlebut,'Enable','off');
set(handles.continuousbut,'Enable','off');
set(handles.savefilebut,'Enable','off');
set(handles.closefilebut,'Enable','off');
set(handles.closebut,'Enable','off');
set(handles.fnameedit,'String','');
set(handles.statustxt,'String','Closed');
set(handles.statustxt,'ForegroundColor',[1 0 0]);
set(handles.fstatustxt,'String','Closed');
set(handles.fstatustxt,'ForegroundColor',[1 0 0]);
set(handles.scansedit,'String','0');
% Update handles structure
guidata(hObject,handles);


% --------------------------------------------------------------------
% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
closefile(hObject,eventdata,handles);
reset_gui(hObject,eventdata,handles);
% Hint: delete(hObject) closes the figure
%DisconnectXY(); %Comment out if no XY stage
delete(hObject);



% --------------------------------------------------------------------
```

```
% --- Executes on button press in savefilebut.
function savefilebut_Callback(hObject, eventdata, handles)
% hObject    handle to savefilebut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
% Get save file name from user
[filename,pathname] = uiputfile('*.txt','Save to TXT File');
filename = [pathname,filename];
% If filename is not valid or user canceled save dialog then return
if(isnumeric(filename))
   return;
end
% Open file for writing and return if it fails
handles.savefile_id = fopen(filename,'w');
if(handles.savefile_id == -1)
   errordlg(['ERROR - Could not open output file: ',filename],'modal')
   return;
end
% Set GUI controls
set(handles.fnameedit,'String',filename);
set(handles.savefilebut,'Enable','off');
set(handles.closefilebut,'Enable','on');
set(handles.scansedit,'String','0');
set(handles.fstatustxt,'String','Open');
set(handles.fstatustxt,'ForegroundColor',[0 1 0]);
% Set saving flag
handles.saving_data = 1;
handles.scans_saved = 0;
guidata(hObject,handles);



% --------------------------------------------------------------------
function fnameedit_Callback(hObject, eventdata, handles)
% hObject    handle to fnameedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of fnameedit as text
%        str2double(get(hObject,'String')) returns contents of fnameedit as a double
% --------------------------------------------------------------------
```

```
% --------------------------------------------------------------------
% --- Executes during object creation, after setting all properties.
function fnameedit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to fnameedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
% --------------------------------------------------------------------
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --------------------------------------------------------------------
% --- Executes on button press in closebut.
function closebut_Callback(hObject, eventdata, handles)
% hObject    handle to closebut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
closefile(hObject, eventdata, handles);
reset_gui(hObject,eventdata,handles);


% --------------------------------------------------------------------
function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double
% --------------------------------------------------------------------



% --------------------------------------------------------------------
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
% --------------------------------------------------------------------
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --------------------------------------------------------------------
function scansedit_Callback(hObject, eventdata, handles)
% hObject    handle to scansedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of scansedit as text
%        str2double(get(hObject,'String')) returns contents of scansedit as a double
% --------------------------------------------------------------------



% --------------------------------------------------------------------
% --- Executes during object creation, after setting all properties.
function scansedit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to scansedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
% --------------------------------------------------------------------
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --------------------------------------------------------------------
% --- Executes on button press in closefilebut.
function closefilebut_Callback(hObject, eventdata, handles)
% hObject    handle to closefilebut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% ------------------------------------------------------------------
closefile(hObject, eventdata, handles);

function closefile(hObject, eventdata, handles)

if (handles.saving_data)

    % Write data to file
    for i=1:handles.scans_saved
        for j=1:handles.num_leds
            count = fprintf(handles.savefile_id,'%1.6f\t',handles.yvals(i,j));
            if(count < 1)
                errordlg('ERROR - Could not write data to file.','modal');
            end
        end
        fprintf(handles.savefile_id,'\n');
    end
    % Close data file
    fclose(handles.savefile_id);
    % Reset GUI Controls
    set(handles.fnameedit,'String','');
    set(handles.savefilebut,'Enable','on');
    set(handles.closefilebut,'Enable','off');
    set(handles.scansedit,'String','0');
    set(handles.fstatustxt,'String','Closed');
    set(handles.fstatustxt,'ForegroundColor',[1 0 0]);
    % Reset saving flag
    handles.saving_data = 0;
    handles.scans_saved = 1;
    guidata(hObject,handles);
end

%Comment out if no XY stage
%function moveNext()
%global MoveArray;
%global posXY;
%global mvArrayLen;

%posXY = posXY + 1;
%posX = MoveArray(posXY,1);
%posY = MoveArray(posXY,2);

%MoveXY(posX, posY);
%pause(1.5);
```

```
%Comment out if no XY stage
%function checkReset()
%global posXY;
%global mvArrayLen;
%reset = 0;
%if (reset)
%    if (posXY >= mvArrayLen)
%        posXY = 0;
%        HomeXY();
%        pause(0.5);
%    else
%        if (rem(posXY, 6) == 0)
%          HomeXY();
%          pause(0.5);
%        end
%    end
%end
```

Function A15 – OQOSSSIgui

```
% ------------------------------------------------------------------
function varargout = OQOSSSIgui(varargin)
% OQOSSSIgui M-file for OQOSSSIgui.fig
%      OQOSSSIgui, by itself, creates a new OQOSSSIgui or raises the existing
%      singleton*.
%
%      H = OQOSSSIgui returns the handle to a new OQOSSSIgui or the handle to
%      the existing singleton*.
%
%      OQOSSSIgui('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in OQOSSSIgui.M with the given input arguments.
%
%      OQOSSSIgui('Property','Value',...) creates a new OQOSSSIgui or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before OQOSSSIgui_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to OQOSSSIgui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help OQOSSSIgui

% Last Modified by GUIDE v2.5 27-Oct-2004 15:23:16
% ------------------------------------------------------------------
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @OQOSSSIgui_OpeningFcn, ...
                   'gui_OutputFcn',  @OQOSSSIgui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --------------------------------------------------------------------
% --- Executes just before OQOSSSIgui is made visible.
function OQOSSSIgui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to OQOSSSIgui (see VARARGIN)
% --------------------------------------------------------------------
% Choose default command line output for OQOSSSIgui
handles.output = hObject;

% Parse initialization file and setup GUI
% Initialization file 'SSSIgui.ini' must be in same directory as fig and m
% files
fid = fopen('SSSIgui.ini','rt');
if(fid == -1)
    error(['ERROR - Could not open GUI initialization file ','OQOSSSIgui.ini']);
end
try
    %Get baud rate for serial communications
    nextline = fgetl(fid);
    if(findstr(nextline,'BAUDRATE') ~= 1)
        error('ERROR - Improper format in initialization file');
    end
    loc = findstr(nextline,'=');
    handles.search_baud = str2num(nextline(loc+1:length(nextline)));
    %Get number of leds
    nextline = fgetl(fid);
    if(findstr(nextline,'HADAMARDLEDS') ~= 1)
        error('ERROR - Improper format in initialization file');
    end
    loc = findstr(nextline,'=');
    handles.num_leds = str2num(nextline(loc+1:length(nextline)));
    %Get number of hadamard sequences
    nextline = fgetl(fid);
    if(findstr(nextline,'HADAMARDSEQS') ~= 1)
        error('ERROR - Improper format in initialization file');
    end
```

```
    loc = findstr(nextline,'=');
    handles.num_seqs = str2num(nextline(loc+1:length(nextline)));
    %Get saturation value for detector system
    nextline = fgetl(fid);
    if(findstr(nextline,'SATURATION') ~= 1)
       error('ERROR - Improper format in initialization file');
    end
    loc = findstr(nextline,'=');
    handles.saturation_value = str2num(nextline(loc+1:length(nextline)));
    %Get x-axis values corresponding to each led
    nextline = fgetl(fid);
    if(findstr(nextline,'XVALS') ~= 1)
       error('ERROR - Improper format in initialization file');
    end
    loc = findstr(nextline,'=');
    handles.xvals = str2num(nextline(loc+1:length(nextline)));
    if(length(handles.xvals) ~= handles.num_leds)
       error('ERROR - Length of XVALS does not match number of LEDs');
    end
    %Get hadamard sequences
    nextline = fgetl(fid);
    if(findstr(nextline,'SEQS') ~= 1)
       error('ERROR - Improper format in initialization file');
    end
    handles.seqs = zeros(handles.num_seqs,handles.num_leds);
    for i=1:handles.num_seqs
       handles.seqs(i,:) = str2num(fgetl(fid));
    end
catch
    fclose(fid);
    error(lasterr,'modal');
end

% Find pseudo-inverse of hadamard transform
handles.hinv = pinv(handles.seqs);

% Initializes flags in handles structure
handles.search_open = 0;
handles.saving_data = 0;
handles.scans_saved = 1;

% Create bar graph handles
axes(handles.axes1);
handles.bar_handles = zeros(5,1);
handles.bar_handles(1) = bar(handles.xvals(1:5),ones(5,1));
```

```
set(handles.bar_handles(1), 'FaceColor',[0 0 1]) %blue
hold;
handles.bar_handles(2) = bar(handles.xvals(6:10),ones(5,1));
set(handles.bar_handles(2), 'FaceColor', [0 1 0]) %green
handles.bar_handles(3) = bar(handles.xvals(11:15),ones(5,1));
set(handles.bar_handles(3), 'FaceColor', [1 1 0]) %yellow
handles.bar_handles(4) = bar(handles.xvals(16:20),ones(5,1));
set(handles.bar_handles(4), 'FaceColor', [1 0.5 0]) %orange
handles.bar_handles(5) = bar(handles.xvals(21:25),ones(5,1));
set(handles.bar_handles(5), 'FaceColor', [1 0 0]) %red

set(gca,'xtick',[1,2,3,4,5]);

% Set GUI control defaults
reset_gui(hObject,eventdata,handles);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes OQOSSSIgui wait for user response (see UIRESUME)
% uiwait(handles.figure1);



% --------------------------------------------------------------------
% --- Outputs from this function are returned to the command line.
function varargout = OQOSSSIgui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
% Get default command line output from handles structure
varargout{1} = handles.output;

% --------------------------------------------------------------------
% --- Executes on button press in singlebut.
function singlebut_Callback(hObject, eventdata, handles)
% hObject    handle to singlebut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------

collect_single(hObject, eventdata, handles);
```

```
% --------------------------------------------------------------------
% --- Executes on button press in continuousbut.
function continuousbut_Callback(hObject, eventdata, handles)
% hObject    handle to continuousbut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
%global mvArrayLen; %Comment out if no XY stage
%global posXY; %Comment out if no XY stage
for i=1:100  % Change the i value in collect_single to match the maximum number of
scans
   if length(get(handles.continuousbut,'Enable')) == 2,
      handles = collect_single(hObject, eventdata, handles);
   else,
      gui_mainfcn(gui_State, varargin{:});
   end
end

% --------------------------------------------------------------------
% --- Collects a single spectrum from the SEARCH instrument
function handles = collect_single(hObject, eventdata, handles)
% --------------------------------------------------------------------
%moveNext(); %Comment out if no XY stage
try
   numscans = 1;
   data = zeros(handles.num_seqs,numscans);
   vals = zeros(handles.num_seqs);

   for n = 1:numscans
      fprintf(handles.search_id,'');
      commtest = 0;
      while(handles.search_id.BytesAvailable < handles.num_seqs*5)
         pause(0.01);
         commtest = commtest + 1;
         if commtest == 500,
            reset_gui(hObject, eventdata, handles)
            errordlg('Communication Error - Please verify that SSSI is connected properly
and turned on.','Communication Error');
            return
            %  figure1_CloseRequestFcn(hObject, eventdata, handles)
         end
      end
      for i=1:handles.num_seqs
         temp = fread(handles.search_id,1,'uint8');
         if(temp ~= i)
```

```
            error('1','ERROR - Invalid sequence order output from SSSI.');
        end
        temp = fread(handles.search_id,1,'uint32');
        data(i,n) = double(temp);
    end
end

% Average
for i = 1:handles.num_seqs
    for n = 1:numscans
        vals(i) = vals(i) + data(i,n);
    end
    vals(i) = vals(i)/numscans;
end
if(handles.search_id.BytesAvailable > 0)
    dummy = fread(handles.search_id,handles.search_id.BytesAvailable);
end
catch
    %errordlg(lasterr,'modal');
    return;
end

if(~isempty(find(vals >= handles.saturation_value)))
    errordlg('ERROR - Signal saturation encountered','modal');
    return;
end

if (handles.saving_data)
    handles.scans_saved = handles.scans_saved + 1;
end

vals = handles.hinv*vals;
for i = 1:handles.num_leds
    if vals(i,1) < 0
        vals(i,1) = 0;
    end
    handles.yvals(handles.scans_saved, i) = vals(i,1);
end

axes(handles.axes1);

for i=1:5
    set(handles.bar_handles(i),'YData',vals(((i-1)*5+1):(i*5)));
end
set(handles.scansedit,'String',num2str(handles.scans_saved));
```

```
guidata(hObject,handles);
%checkReset(); %Comment out if no XY stage


% --------------------------------------------------------------------
% --- Executes on button press in openbut.
function openbut_Callback(hObject, eventdata, handles)
% hObject    handle to openbut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
% Reset GUI
reset_gui(hObject,eventdata,handles);
% Get specified serial port from popup menu
port_contents = get(handles.portpop,'String');
handles.search_port = port_contents{get(handles.portpop,'Value')};
% Create serial object and open communications
handles.search_id =
serial(handles.search_port,'BaudRate',handles.search_baud,'Timeout',10);
try
   fopen(handles.search_id);
catch
   errordlg(['ERROR - Could not open communications on ',handles.search_port, '.
Matlab returned error: ',lasterr],'modal');
   delete(handles.search_id);
   return;
end

handles.search_open = 1;

% Test communications to see if SSSI is connected

numscans = 1;
data = zeros(handles.num_seqs,numscans);
vals = zeros(handles.num_seqs);

for n = 1:numscans
   fprintf(handles.search_id,'');
   commtest = 0;
   while(handles.search_id.BytesAvailable < handles.num_seqs*5)
      pause(0.01);
      commtest = commtest + 1;
      if commtest == 500,
         reset_gui(hObject, eventdata, handles)
```

```
            errordlg('Communication Error - Please verify that SSSI is connected properly
and turned on.','Communication Error');
            return
            %  figure1_CloseRequestFcn(hObject, eventdata, handles)
        end
    end
    for i=1:handles.num_seqs
        temp = fread(handles.search_id,1,'uint8');
        if(temp ~= i)
            error('1','ERROR - Invalid sequence order output from SSSI.');
        end
        temp = fread(handles.search_id,1,'uint32');
        data(i,n) = double(temp);
    end
end

% Average
for i = 1:handles.num_seqs
    for n = 1:numscans
        vals(i) = vals(i) + data(i,n);
    end
    vals(i) = vals(i)/numscans;
end
if(handles.search_id.BytesAvailable > 0)
    dummy = fread(handles.search_id,handles.search_id.BytesAvailable);
end



    %Comment out if no XY stage
    % Initialize XY-Stage
    %InitXY('COM4');
    %global MoveArray;
    %global posXY;
    %global mvArrayLen;
    %posXY = 0;
    %MoveArray = load('MoveArray.txt');
    %[mvArrayLen, tmp] = size(MoveArray);
    %clear tmp;
    %handles.yvals = zeros(mvArrayLen, handles.num_seqs);

    % Enable necessary GUI controls
    set(handles.singlebut,'Enable','on');
    set(handles.continuousbut,'Enable','on');
    set(handles.savefilebut,'Enable','on');
```

```
    set(handles.closebut,'Enable','on');
    set(handles.statustxt,'String','Open');
    set(handles.statustxt,'ForegroundColor',[0 1 0]);
    % Update handles structure
    guidata(hObject,handles);



% --------------------------------------------------------------------
% --- Executes on selection change in portpop.
function portpop_Callback(hObject, eventdata, handles)
% hObject    handle to portpop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns portpop contents as cell array
%        contents{get(hObject,'Value')} returns selected item from portpop
% --------------------------------------------------------------------



% --------------------------------------------------------------------
% --- Executes during object creation, after setting all properties.
function portpop_CreateFcn(hObject, eventdata, handles)
% hObject    handle to portpop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
% --------------------------------------------------------------------
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --------------------------------------------------------------------
% --- Reset GUI variables and controls
function reset_gui(hObject, eventdata, handles)
% --------------------------------------------------------------------
% Check if search is open and close
if(handles.search_open)
    fclose(handles.search_id);
    delete(handles.search_id);
```

```
    %HomeXY(); %Comment out if no XY stage
    %DisconnectXY(); %Comment out if no XY stage
end
if(handles.saving_data)
    fclose(handles.savefile_id);
    delete(handles.savefile_id);
end
handles.search_id = [];
handles.savefile_id = [];
handles.search_open = 0;
handles.saving_data = 0;
handles.scans_saved = 1;
% Set GUI control defaults
set(handles.singlebut,'Enable','off');
set(handles.continuousbut,'Enable','off');
set(handles.savefilebut,'Enable','off');
set(handles.closefilebut,'Enable','off');
set(handles.closebut,'Enable','off');
set(handles.fnameedit,'String','');
set(handles.statustxt,'String','Closed');
set(handles.statustxt,'ForegroundColor',[1 0 0]);
set(handles.fstatustxt,'String','Closed');
set(handles.fstatustxt,'ForegroundColor',[1 0 0]);
set(handles.scansedit,'String','0');
% Update handles structure
guidata(hObject,handles);


% --------------------------------------------------------------------
% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
closefile(hObject,eventdata,handles);
reset_gui(hObject,eventdata,handles);
% Hint: delete(hObject) closes the figure
%DisconnectXY(); %Comment out if no XY stage
delete(hObject);



% --------------------------------------------------------------------
% --- Executes on button press in savefilebut.
```

```
function savefilebut_Callback(hObject, eventdata, handles)
% hObject    handle to savefilebut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% ------------------------------------------------------------------
% Get save file name from user
[filename,pathname] = uiputfile('*.txt','Save to TXT File');
filename = [pathname,filename];
% If filename is not valid or user canceled save dialog then return
if(isnumeric(filename))
   return;
end
% Open file for writing and return if it fails
handles.savefile_id = fopen(filename,'w');
if(handles.savefile_id == -1)
   errordlg(['ERROR - Could not open output file: ',filename],'modal')
   return;
end
% Set GUI controls
set(handles.fnameedit,'String',filename);
set(handles.savefilebut,'Enable','off');
set(handles.closefilebut,'Enable','on');
set(handles.scansedit,'String','0');
set(handles.fstatustxt,'String','Open');
set(handles.fstatustxt,'ForegroundColor',[0 1 0]);
% Set saving flag
handles.saving_data = 1;
handles.scans_saved = 0;
guidata(hObject,handles);




% ------------------------------------------------------------------
function fnameedit_Callback(hObject, eventdata, handles)
% hObject    handle to fnameedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of fnameedit as text
%        str2double(get(hObject,'String')) returns contents of fnameedit as a double
% ------------------------------------------------------------------



% ------------------------------------------------------------------
```

```
% --- Executes during object creation, after setting all properties.
function fnameedit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to fnameedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
% --------------------------------------------------------------------
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --------------------------------------------------------------------
% --- Executes on button press in closebut.
function closebut_Callback(hObject, eventdata, handles)
% hObject    handle to closebut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
closefile(hObject, eventdata, handles);
reset_gui(hObject,eventdata,handles);


% --------------------------------------------------------------------
function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double
% --------------------------------------------------------------------



% --------------------------------------------------------------------
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
```

```
%       See ISPC and COMPUTER.
% --------------------------------------------------------------------
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --------------------------------------------------------------------
function scansedit_Callback(hObject, eventdata, handles)
% hObject    handle to scansedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of scansedit as text
%        str2double(get(hObject,'String')) returns contents of scansedit as a double
% --------------------------------------------------------------------



% --------------------------------------------------------------------
% --- Executes during object creation, after setting all properties.
function scansedit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to scansedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
% --------------------------------------------------------------------
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --------------------------------------------------------------------
% --- Executes on button press in closefilebut.
function closefilebut_Callback(hObject, eventdata, handles)
% hObject    handle to closefilebut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --------------------------------------------------------------------
closefile(hObject, eventdata, handles);

function closefile(hObject, eventdata, handles)

if (handles.saving_data)

   % Write data to file
   for i=1:handles.scans_saved
      for j=1:handles.num_leds
         count = fprintf(handles.savefile_id,'%1.6f\t',handles.yvals(i,j));
         if(count < 1)
            errordlg('ERROR - Could not write data to file.','modal');
         end
      end
      fprintf(handles.savefile_id,'\n');
   end
   % Close data file
   fclose(handles.savefile_id);
   % Reset GUI Controls
   set(handles.fnameedit,'String','');
   set(handles.savefilebut,'Enable','on');
   set(handles.closefilebut,'Enable','off');
   set(handles.scansedit,'String','0');
   set(handles.fstatustxt,'String','Closed');
   set(handles.fstatustxt,'ForegroundColor',[1 0 0]);
   % Reset saving flag
   handles.saving_data = 0;
   handles.scans_saved = 1;
   guidata(hObject,handles);
end

%Comment out if no XY stage
%function moveNext()
%global MoveArray;
%global posXY;
%global mvArrayLen;

%posXY = posXY + 1;
%posX = MoveArray(posXY,1);
%posY = MoveArray(posXY,2);

%MoveXY(posX, posY);
%pause(1.5);
```

```
%Comment out if no XY stage
%function checkReset()
%global posXY;
%global mvArrayLen;
%reset = 0;
%if (reset)
%    if (posXY >= mvArrayLen)
%        posXY = 0;
%        HomeXY();
%        pause(0.5);
%    else
%        if (rem(posXY, 6) == 0)
%          HomeXY();
%          pause(0.5);
%        end
%    end
%end
```

Function A16 – ResPlot

```
function ResPlot(file)
resdata = load(file);
[a,b] = size(resdata);
% Normalize the data
 for j=1:b
    largest = 0;
    for i=1:a
       if resdata(i,j) > largest
          largest = resdata(i,j);
       end
    end
    resdata(1:a, j) = resdata(1:a, j)/largest;
 end

%after the normalization, this shifts the data so each LED in the rows to line up with all
others
%in that row
% By: Clay Harris
for c = 1:5
   for d = 1:5
      resdata (:,(((c-1)*5)+ d)) = circshift(resdata(:,(((c-1)*5)+ d)),[-6*(d-1),0]);
      resdata ((a+((d-1)*(-6))):a,(((c-1)*5)+ d)) = 1;
   end
end
% Clay's Loop Ends Here!!

% plots each row of LED's at once
% for i=1:5
%     figure;
%    plot(resdata(:,(i-1)*5+1:i*5),'k');
% end

% plots each LED at once
% for i = 1:25
%   figure;
%    resdata = resdata/1000;
   plot(resdata(1:70,1),'b');

%     pause;
% end
```

Function A17 – GroupColors

```
function GroupColors
colorfile = load(file);
colordata = zeros(100,25,6);
blue = 1;
red = 2;
green = 3;
yellow = 4;
white = 5;
orange = 6;
for i = 1:100
   for j = 1:25
      colordata(i,j,blue) = colorfile(6*(i-1)+blue,j);
      colordata(i,j,red) = colorfile(6*(i-1)+red,j);
      colordata(i,j,green) = colorfile(6*(i-1)+green,j);
      colordata(i,j,yellow) = colorfile(6*(i-1)+yellow,j);
      colordata(i,j,white) = colorfile(6*(i-1)+white,j);
      colordata(i,j,orange) = colorfile(6*(i-1)+orange,j);
   end
end
```

Function A18 – InitXY

```
function result = InitXY(port)
global serialObj;
global init;
global homed;
global maxSteps;
global safePos;
homed = 0;

if (isempty(init)) || (init == 0)
   serialObj = serial(port);
   set(serialObj,'BaudRate',115200,'Terminator','CR','Timeout',10);
   fopen(serialObj);
end

init = 1;

% P is the "Power" setting command...there are three arguments
% separated by commas...the first is the total current setting, the
% second is the Idle current setting, the third is the decay rate of the
% idle current setting...this is beyond the scope of a comment, please
% refer to the manual starting on page 38.

% the maximum current setting = 102.4*maximum motor current...The maximum
% setting is 2.5 A which is a current setting of 256.

result = SendCommand('X0P123,30,0');
if IsXYError(result)
   return
end

result = SendCommand('Y0P123,30,0');
if IsXYError(result)
   return
end

% Step mode (The maximum limits MUST be changed if this value is
% changed!!)...0 = Full step mode (1800 maximum limit), 1 = Half step
% mode (default, 3600 max limit), 2 = Quarter step mode (7200 max limit),
% 3 = Eighth step mode (14400 max limit), 4 = Sixteenth step mode
% (28800 max limit).
stepMode = 1;
maxSteps = 1800 .* 2.^stepMode;
result = SendCommand(strcat('X0H', int2str(stepMode)));
```

```
if IsXYError(result)
   return
end

result = SendCommand(strcat('Y0H', int2str(stepMode)));
if IsXYError(result)
   return
end

% B is the "Beginning" Velocity or Initial/Final Velocity...If the motor
% movement stalls upon first attempting to move, decrease B in 50 unit
% increments

begSpeed = 30 .* 2.^stepMode;
result = SendCommand(strcat('X0B', int2str(begSpeed)));
if IsXYError(result)
   return
end

result = SendCommand(strcat('Y0B', int2str(begSpeed)));
if IsXYError(result)
   return
end

% E is the "End" Velocity or Maximum Velocity...If motor movement stalls
% at top velocity, decrease E in 100 unit increments
endSpeed = 10 .* begSpeed;
result = SendCommand(strcat('X0E', int2str(endSpeed)));
if IsXYError(result)
   return
end

result = SendCommand(strcat('Y0E', int2str(endSpeed)));
if IsXYError(result)
   return
end

% S is the Slope of the Ramp at both the beginning and end of
% movement...The number of steps to reach maximum velocity = (E-B)/S
stepSpeed = 0;
result = SendCommand(strcat('X0S', int2str(stepSpeed)));
if IsXYError(result)
   return
end
```

```
result = SendCommand(strcat('Y0S', int2str(stepSpeed)));
if IsXYError(result)
    return
end

% Go Home
safePos = 25 .* 2.^stepMode;
result = HomeXY();
if IsXYError(result)
    return
end
homed = 1;
```

Function A19 – test

```
result = InitXY('Com4');
if IsXYError(result)
   sprintf ('XY-Stage Error" detected during initialization')
   DisconnectXY();
   return
end
MoveArray = load('MoveArray.txt');
MoveXYArray(MoveArray);
sprintf('Press a key to reset to Home')
pause;
HomeXY();
sprintf('When the platform comes to rest, press any key to end')
pause;
DisconnectXY();
clear result;
```

Function A20 – TestSpeed

```
global maxSteps;
result = InitXY('COM4');
if IsXYError(result)
    sprintf(result)
    DisconnectXY();
else
    pause;
    MoveXY(maxSteps, maxSteps);
    pause;
    HomeXY();
    DisconnectXY();
end
```

Function A21 – SendCommand

```
function result = SendCommand(command)
global serialObj;

    fprintf(serialObj,command)
    result = GetResult();
```

Function A22 – XYControl

```
s = serial('COM4');
set(s,'BaudRate',115200,'Terminator','CR','Timeout',1);
fopen(s);
fprintf(s,'X0P123,0,0')
out = fscanf(s,'%s')
fprintf(s,'X0E2000')
out = fscanf(s)
fprintf(s,'X0B300')
out = fscanf(s)
fprintf(s,'X0S2')
out = fscanf(s)
fprintf(s,'X0N+1')
out = fscanf(s)
fprintf(s,'X0 max 3600')
out = fscanf(s)
fprintf(s,'Y0P123,0,0')
out = fscanf(s)
fprintf(s,'Y0E2000')
out = fscanf(s)
fprintf(s,'Y0B300')
out = fscanf(s)
fprintf(s,'Y0S2')
out = fscanf(s)
fprintf(s,'Y0N+1')
out = fscanf(s)
fprintf(s,'Y0 max 3600')
out = fscanf(s)
fclose(s)
delete(s)
clear s
```

Function A23 – MoveXY

```
function result = MoveXY(x, y)
global maxSteps;
if (x < 0)
   result = 'X is less than 0, moving to 0 instead.'
   x = 0;
end
if (x > maxSteps)
   result = 'X exceeds maxSteps, moving to maxSteps instead.'
   x = maxSteps;
end
if (y < 0)
   result = 'Y is less than 0, moving to 0 instead.'
   y = 0;
end
if (y > maxSteps)
   result = 'Y exceeds maxSteps, moving to maxSteps instead.'
   y = maxSteps;
end
MovX = strcat('X0M', int2str(x));
MovY = strcat('Y0M', int2str(y));
result = SendCommand(MovX);
if IsXYError(result)
   return
end
result = SendCommand(MovY);
if IsXYError(result)
   return
end

[X, Y] = ReadPosXY();
while ((X ~= x) || (Y ~= y))
   [X, Y] = ReadPosXY();
end
```

Function A24 – MoveXYArray

```
function result = MoveXYArray(MoveArray)
[a,b] = size(MoveArray);
if b == 2
   for i = 1:1:a
      x = MoveArray(i,1);
      y = MoveArray(i,2);
      result = MoveXY(x,y);
   end
end
```

Function A25 – ReadPosXY

```
function [x, y] = ReadPosXY()
posX = SendCommand('X0m');
posY = SendCommand('Y0m');
if((numel(posX) < 14) || (numel(posY) < 14))
   DisconnectXY();
end
x = str2num(posX(4:numel(posX)-1));
y = str2num(posY(4:numel(posY)-1));
if((~isnumeric(x)) || (~isnumeric(y)))
   x = 0;
   y = 0;
   DisconnectXY();
end
```

Function A26 – IsXYError

```
function badResult = IsXYError(result)
   if (strncmp('x0>',result,3) || strncmp('y0>',result,3) || strncmp('x0b',result,3) ||
strncmp('y0b',result,3))
      badResult = 0;
      return
   else
      sprintf(result);
      badResult = 1;
   end
```

Function A27 – GetResult

```
function result = GetResult()
global serialObj;

   result = fscanf(serialObj);
```

Function A28 – KillPowXY

```
function KillPowXY()
SendCommand('X0P1,0,0');
SendCommand('Y0P1,0,0');
```

Function A29 – DisconnectXY

```
function DisconnectXY()
global serialObj;
global init;
if init == 1
   init = 0;
   HomeXY();
   pause(4);
   KillPowXY();
   pause(0.5);
   fclose(serialObj);
   delete(serialObj);
   clear global serialObj;
end
```

Function A30 – PCAColorData

```
function PCAColorData(file)
colordata = load(file);
colordatamc = colordata - repmat(mean(colordata),600,1);
% colordatamc = zscore(colordat);
[scores,loads,ssq,res,reslm,tsqlm,tsq] = pca(colordatamc,0,6,24);
plot3 (scores(:,1),scores(:,2),scores(:,3),'k.');

xlabel('PC1');
ylabel('PC2');
zlabel('PC3');

ssqtable(ssq, 24);


% R = zeros(24);
% for i = 1:6
%     R(((i-1)*100+1):i*100) = i;
% end

R = [1,2,3,4,5,6,1,2,3,4,5,6,1,2,3,4,5,6,1,2,3,4,5,6];


pcaVar = [61.85, 32.05, 5.66];
pcellipse(scores, R, 50, pcaVar, [1,2,3]);

% data = principal component scores
% R = concentration or group vector
% nosd = number of standard deviations to include
% s = % of the variance from pca
% take = pcs to include in the model, for largest separation use take =
% [1,2,3]
```

Near-field Scanning Optical Microscope Functions

Function A31 – SoundRecorder

```
function varargout = SoundRecorder(varargin)
% SOUNDRECORDER GUI is meant to serve as an example for MATLAB users in the
Test & Measurement area – Heavily edited by Justin Clay Harris 2005
%
%    SOUNDRECORDER continuously displays data to 2 different axes during a data
%    acquisition session, allows the user to input parameters during the
%    session, allows the user to play back the acquired signal, allows the user
%    to save and/or export the acquired signal, allows the user to pan through the
%    acquired signal, and uses graphics on the GUI components to create a more
%    aesthetically appealing application
%
%    The products required to run SOUNDRECORDER are:
%       MATLAB 6.5 (R13)
%       Data Acquisition Toolbox 2.2
%
%    SOUNDRECORDER contains the following components:
%       Push Button
%       Axis
%       Text Field
%       Frame
%       Edit Box
%       Toggle Button
%       Menu
%       Slider
%
%    See also DAQ2AXIS, DAQ2AXISFIELD, DAQ2AXISFIELDPLAY.

% Edit the above text to modify the response to help SoundRecorder

% Last Modified by GUIDE v2.5 13-Aug-2003 23:08:45

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
   'gui_Singleton',  gui_Singleton, ...
   'gui_OpeningFcn', @SoundRecorder_OpeningFcn, ...
   'gui_OutputFcn',  @SoundRecorder_OutputFcn, ...
   'gui_LayoutFcn',  [], ...
   'gui_Callback',   []);
if nargin & isstr(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
```

```
end

if nargout
   varargout{1:nargout} = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before SoundRecorder is made visible.
function SoundRecorder_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for SoundRecorder
handles.output = hObject;

% Make sure that the DAQ toolbox is installed. If not, then return
if ~exist([matlabroot '\toolbox\daq'],'dir')
   errordlg('You must have the Data Acquisition Toolbox installed to run this
demo','Error','modal');
   return
end

% Set the icons for the pushbuttons
set(handles.start_stop,'CData',recordbutton);
set(handles.play,'CData',playbutton(1),'Enable','off');

try
   % Create an analoginput object using the winsound adaptor
   daq_object = analoginput('winsound');
   chan = addchannel(daq_object,[1 2]);
   dinfo = daqhwinfo(daq_object);

   % Create an analogoutput object using the winsound adaptor
   daq_object_out = analogoutput('winsound');
   chan_out = addchannel(daq_object_out,[1:2]);
catch
   errordlg(['Error while creating ANALOGINPUT/ANALOGOUTPUT Winsound
object',sprintf('\n\n'),...
```

```
        lasterr],'Error','modal');
    return
end

% Preliminary GUI settings
set(hObject,'Renderer','ZBuffer','DoubleBuffer','on','BackingStore','on');
set(handles.send_to_workspace,'Enable','off');
set(handles.send_to_sptool,'Enable','off');
set(handles.send_to_figure,'Enable','off');

if isempty(which('sptool'))
    set(handles.send_to_sptool,'Visible','off');
end

% Initialize the data in the GUI fields
fs = 44100;
dur = 10;
fname = 'untitled';
set(handles.samplerate,'String',num2str(fs));
set(handles.duration,'String',num2str(dur));

% Initialize the axes for the data
axes(handles.chan1);
set(handles.chan1, 'TickLength', [0 0], 'XTickLabel','');
ylabel('Left (V)');
title('Current Signal');

axes(handles.chan2);
set(handles.chan2, 'TickLength', [0 0], 'XTickLabel','');
ylabel('Right (V)');
xlabel('Time (sec)');

axes(handles.sig);
set(handles.sig, 'TickLength', [0 0], 'XTickLabel','');
xlabel('Time (sec)');
title('Full Signal');

axes(handles.freq);
set(handles.freq, 'TickLength', [0 0], 'XTickLabel','');
title('FFT');

axes(handles.freqL);
set(handles.freqL, 'TickLength', [0 0], 'XTickLabel','');
xlabel('Frequency (Hz)');
```

```
% Update handles structure
handles.daq_object = daq_object;
handles.minSampleRate = dinfo.MinSampleRate;
handles.maxSampleRate = dinfo.MaxSampleRate;
handles.daq_object_out = daq_object_out;
handles.fs = fs;
handles.dur = dur;
handles.fname = fname;
guidata(hObject, handles);

% Set some properties of the analoginput DAQ object
set(handles.daq_object,'LoggingMode','disk','LogFileName',[handles.fname '.daq'],...
    'StartFcn',{@start_daq,handles},'StopFcn',{@stop_daq,handles});

% Set some properties of the analogoutput DAQ object
set(daq_object_out,'StartFcn',{@start_daq_out,handles},'StopFcn',{@stop_daq_out,handl
es});

% UIWAIT makes SoundRecorder wait for user response (see UIRESUME)
% uiwait(hObject);


% --- Outputs from this function are returned to the command line.
function varargout = SoundRecorder_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function samplerate_CreateFcn(hObject, eventdata, handles)
% hObject    handle to samplerate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
```

```
end


function samplerate_Callback(hObject, eventdata, handles)
% hObject    handle to samplerate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of samplerate as text
%        str2double(get(hObject,'String')) returns contents of samplerate as a double

% Get the sample rate from the GUI and make sure it is a numeric value
fs = str2num(get(hObject,'String'));
if isempty(fs) | isnan(fs)
   errordlg('Please enter a valid numeric value for the sample rate.','Error','modal');
   set(hObject, 'String', get(handles.daq_object, 'SampleRate'));
   return
end

% Make sure the numeric sample rate provided is a valid value for the sound card
if fs < handles.minSampleRate
   fs = handles.minSampleRate;
   handles.daq_object.samplerate=fs;
   set(hObject, 'String', num2str(fs));
elseif fs > handles.maxSampleRate
   fs = handles.maxSampleRate;
   handles.daq_object.samplerate=fs;
   set(hObject, 'String', num2str(fs));
else
   % Set the Samplerate property of the DAQ object and let the user know
   % what the true sample rate is
   fs = setverify(handles.daq_object,'SampleRate', fs);
   set(hObject, 'String', num2str(fs));
end

% Update the GUI's handles structure
handles.fs = fs;
guidata(hObject,handles);


% --- Executes during object creation, after setting all properties.
function duration_CreateFcn(hObject, eventdata, handles)
% hObject    handle to duration (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function duration_Callback(hObject, eventdata, handles)
% hObject    handle to duration (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of duration as text
%        str2double(get(hObject,'String')) returns contents of duration as a double

% Get the duration from the GUI
dur = get(hObject,'String');

% If the duration is specified as 'inf' or not specified at all, treat as INF
if isempty(dur) | strcmp(lower(dur),'inf')
   dur = inf;
   handles.dur = dur;
   guidata(hObject,handles);
   return
end

% Make sure the duration is a valid numeric value
dur = str2num(dur);
if isempty(dur) | dur < 1 | isnan(dur)
   errordlg('Please enter a valid duration of at least 1 sec.','Error','modal');
   set(hObject,'String','');
   dur = inf;
   handles.dur = dur;
   guidata(hObject,handles);
else
   handles.dur = dur;
   guidata(hObject,handles);
end


% --- Executes on button press in close.
function close_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to close (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% If the winsound object is running, stop it then delete and clear it
if(strcmp(handles.daq_object.running,'On'))
   stop(handles.daq_object);
end
if(strcmp(handles.daq_object_out.running,'On'))
   stop(handles.daq_object_out);
end
delete(handles.daq_object);
delete(handles.daq_object_out);
clear handles.daq_object handles.daq_object_out;

if exist([handles.fname '.daq'])
   delete([handles.fname '.daq']);
end

% close the GUI
delete(handles.figure1)



% --- Executes during object creation, after setting all properties.
function chan1slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to chan1slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --- Executes on slider movement.
function chan1slider_Callback(hObject, eventdata, handles)
% hObject    handle to chan1slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

if strcmp(handles.daq_object.running,'On') ||
strcmp(handles.daq_object_out.running,'On'), return; end

try
   % Identify the data
   data = handles.data;
   fs = handles.fs;
   slider_time = get(hObject,'Value');

   % Make sure that the selected area is within bounds of the data.
   % Extract the local x- and y-data for the preview axes
   if slider_time == 0, slider_time = 1/fs; end
   ydata1 = data(ceil(slider_time*fs):ceil(slider_time*fs+fs/2-1),1);
   ydata2 = data(ceil(slider_time*fs):ceil(slider_time*fs+fs/2-1),2);
   xdata = linspace(slider_time, slider_time+.5, length(ydata1));

   % Move the patch and all line objects to the selected x-position
   set(handles.patch_handle,'XData',[slider_time slider_time+.5 slider_time+.5
slider_time]);
   set(handles.plot1_line_handle,'XData',[slider_time+.25 slider_time+.25]);
   set(handles.plot2_line_handle,'XData',[slider_time+.25 slider_time+.25]);
   set(handles.plot3_line_handle,'XData',[slider_time+.25 slider_time+.25]);

   % Update the Left channel preview axis
   axes(handles.chan1);
   set(handles.plot1_handle,'YData',ydata1,'XData',xdata);
   axis([min(xdata) max(xdata) -1 1]);

   % Update the Right channel preview axis
   axes(handles.chan2);
   set(handles.plot2_handle,'YData',ydata2,'XData',xdata);
   axis([min(xdata) max(xdata) -1 1]);

   % Update frequency axes
   freqstep = (fs/length(ydata1));
   freqvals = 0:freqstep:fs/2;
   Y1 = fft(ydata1);
   Y2 = fft(ydata2);
   Pyy1 = (Y1.* conj(Y1)) / length(ydata1);
   Pyy2 = (Y2.* conj(Y2)) / length(ydata2);
   axes(handles.freq);
```

```
    %set(handles.plotf_handle,'Xdata',freqvals);
    set(handles.plotf_handle,'Ydata',Pyy1(1:(length(ydata1)/2+1)));
    axis([freqvals(1) freqvals(length(freqvals)) 0 max(Pyy1(2:length(Pyy1)))]);
    axes(handles.freqL);
    %set(handles.plotf_handle,'Xdata',freqvals);
    set(handles.plotfL_handle,'Ydata',Pyy2(1:(length(ydata2)/2+1)));
    axis([freqvals(1) freqvals(length(freqvals)) 0 max(Pyy2(2:length(Pyy2)))]);

    drawnow;

catch
    % If there were any errors, display the last error to the screen
    errordlg(lasterr,'Error','modal');
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  MENU
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% --------------------------------------------------------------------
function SRD_help_Callback(hObject, eventdata, handles)
% hObject    handle to SRD_help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Display the HTML documentation for SoundRecorder in the help browser
a = which('soundrecorderdemodoc.html');
if ~isempty(a)
    web(a);
else
    errordlg('Could not find SoundRecorderDemoDoc.html','Error','modal');
end


% --------------------------------------------------------------------
function daq_help_Callback(hObject, eventdata, handles)
% hObject    handle to daq_help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Display the HTML documentation for DAQ in the help browser
doc daq;


% --------------------------------------------------------------------
```

```
function demos_help_Callback(hObject, eventdata, handles)
% hObject    handle to demos_help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Display a list of the available DAQ demos in the help browser
doc daqdemos;


% --------------------------------------------------------------------
function save_as_Callback(hObject, eventdata, handles)
% hObject    handle to save_as (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% ERROR HANDLING
if ~exist([handles.fname '.daq'])
   errordlg('Please acquire some data first before saving','Error','modal');
   return;
end

% Define some parameters that will be used by the SAVE dialog box
filterspec = {'*.daq','DAQ files (*.daq)';'*.mat','MAT files (*.mat)'; '*.wav','WAV files
(*.wav)'};
title = 'Save As';
temp_file = 'untitled';

[filename, path, index] = uiputfile(filterspec, title, temp_file);

% Clean up the filename
if strfind(filename,'.daq'), filename = filename(1:end-4); end
if strfind(filename,'.mat'), filename = filename(1:end-4); end
if strfind(filename,'.wav'), filename = filename(1:end-4); end

data = handles.data;
time = handles.time;
fs = handles.fs;

if index == 0
   % No action taken...the user has cancelled the Save
elseif index == 1
   % Rename the .daq file to the user's choice
   copyfile([handles.fname '.daq'],[path filename '.daq']);
elseif index == 2
   % Save the data as a .mat file
```

```
    save([path filename '.mat'],'data','time','fs');
elseif index == 3
    % Save the data as a .wav file
    warning off
    wavwrite(data,fs,[path filename '.wav']);
    warning on
end


% --------------------------------------------------------------------
function send_to_workspace_Callback(hObject, eventdata, handles)
% hObject    handle to send_to_workspace (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Load the currently used .daq file into the workspace
evalin('base',['[data time] = daqread("', handles.fname, '.daq");']);
evalin('base',['fs = ', num2str(handles.fs), ';']);


% --------------------------------------------------------------------
function send_to_figure_Callback(hObject, eventdata, handles)
% hObject    handle to send_to_figure (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Send the acquired data to a new figure window
f = figure;
data = handles.data;
time = handles.time;

subplot(2,1,1);
plot(time(1:20:end),data(1:20:end,1),'r');
title('SoundRecorder Acquired Signal');
ylabel('Left Channel (V)');
axis([min(time) max(time) -5 5]);

subplot(2,1,2);
plot(time(1:20:end),data(1:20:end,2));
ylabel('Right Channel (V)');
xlabel('Time (sec)');
axis([min(time) max(time) -1 1]);


% --------------------------------------------------------------------
```

```
function send_to_sptool_Callback(hObject, eventdata, handles)
% hObject    handle to send_to_sptool (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Send the acquired data to SPTool
try
   sptool('load','Signal',handles.data,handles.fs,'SoundRecorderData');
catch
   errordlg('Failed sending acquired data to SPTool','Error','Modal');
end



% --- Executed when the user pushes the Record\Stop toggle button
function start_stop_Callback(hObject, eventdata, handles)
% hObject    handle to start_stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Update the handles structure
handles = guidata(handles.figure1);


% Get the parameters supplied by the user and update the GUI handles
% structure
fname = handles.fname;
fs = handles.fs;
dur = handles.dur;

% Define a preview of the signal to be 1/10 a second of data
preview = ceil(fs/10);

freqstep = (fs/preview);
freqvals = 0:freqstep:fs/2;

if get(hObject,'Value') == 1  % Start data collection

   % Set the winsound object general properties
   set(handles.daq_object,'SampleRate',fs,'LoggingMode','disk','LogFileName',[fname
'.daq']);

set(handles.daq_object_out,'SampleRate',fs,'SamplesOutputFcnCount',preview,'Samples
OutputFcn',{@selectRegion,handles,2});
```

```
    % Initialize the preview axes
    axes(handles.chan1);
    plot1_handle = plot(zeros(1,preview),'r-');
    axis tight;
    set(handles.chan1, 'TickLength', [0 0], 'XTickLabel','');
    ylabel('Left (V)');
    title('Current Signal');

    axes(handles.chan2);
    plot2_handle = plot(zeros(1,preview),'b-');
    axis tight;
    set(handles.chan2, 'TickLength', [0 0], 'XTickLabel','');
    ylabel('Right (V)');
    xlabel('Time (sec)');

    % Initialize the frequency axes
    axes(handles.freq);
    plotf_handle = plot(freqvals,zeros(1,length(freqvals)),'r-');
    %axis tight;
    %set(handles.freq, 'TickLength', [0.01 0]);
    ylabel('Power');
    title('FFT');
    axes(handles.freqL);
    plotfL_handle = plot(freqvals,zeros(1,length(freqvals)),'b-');
    %axis tight;
    %set(handles.freq, 'TickLength', [0.01 0]);
    ylabel('Power');
    xlabel('frequency');

    % If the duration is infinite, record indefinately
    if ~isfinite(dur)
        set(handles.daq_object,'SamplesPerTrigger',inf,'TriggerType','immediate');

        % Start recording and abort if there is an error
        if ~start_acquisition(handles)
            return;
        end

        % Update the preview axes
        update_plots(plot1_handle, plot2_handle, plotf_handle, plotfL_handle,handles,
preview);

        % Stop recording and abort if there is an error
        if ~stop_acquisition(handles)
            return;
```

```
        end

        % If a duration is specified, record to the maximum time allowed
    else
        set(handles.daq_object,'SamplesPerTrigger',fs*dur,'TriggerType','immediate');

        % Start recording and abort if there is an error
        if ~start_acquisition(handles)
            return;
        end

        % Update the preview axes
        update_plots(plot1_handle, plot2_handle, plotf_handle, plotfL_handle,handles,
preview);

        % Stop recording and abort if there is an error
        if ~stop_acquisition(handles)
            return;
        end

    end

else  % Stop data collection
    if ~stop_acquisition(handles)
        evalin('base',['[data time] = daqread("', handles.fname, '.daq");']);
        evalin('base',['fs = ', num2str(handles.fs), ';']);
        return;
    end
end

% Load the data into the GUI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[data time] = daqread([fname '.daq']);

% Update the Left channel preview axis
axes(handles.chan1);
plot1_handle = plot(time(1:ceil(fs/2),1), data(1:ceil(fs/2),1), 'r-');
set(handles.chan1, 'TickLength', [0 0], 'XTickLabel','');
ylabel('Left (V)');
title('Current Signal');
plot1_line_handle = line([time(ceil(fs/4)) time(ceil(fs/4))],[-1 1]);
set(plot1_line_handle,'color',[1 0 1]);

% Update the Right channel preview axis
```

```
axes(handles.chan2);
plot2_handle = plot(time(1:ceil(fs/2),1), data(1:ceil(fs/2),2), 'b-');
set(handles.chan2, 'TickLength', [0 0]);
ylabel('Right (V)');
xlabel('Time (sec)');
plot2_line_handle = line([time(ceil(fs/4)) time(ceil(fs/4))],[-1 1]);
set(plot2_line_handle,'color',[1 0 1]);

% Update the frequency axes
freqstep = (fs/ceil(fs/2));
freqvals = 0:freqstep:fs/2;
Y1 = fft(data(1:ceil(fs/2),1));
Y2 = fft(data(1:ceil(fs/2),2));
Pyy1 = (Y1.* conj(Y1)) / ceil(fs/2);
Pyy2 = (Y2.* conj(Y2)) / ceil(fs/2);
axes(handles.freq);
plotf_handle = plot(freqvals,Pyy1(1:ceil(fs/2)/2+1),'r-');
axis tight;
set(handles.freq, 'TickLength', [0.01 0]);
ylabel('Power');
title('FFT');
axes(handles.freqL);
plotfL_handle = plot(freqvals,Pyy2(1:ceil(fs/2)/2+1),'b-');
axis tight;
set(handles.freq, 'TickLength', [0.01 0]);
ylabel('Power');
xlabel('frequency');

% Define the slider properties based on the acquired data
set(handles.chan1slider,'Value',1/fs,'min',0,'max',max(time)-.5, 'sliderstep',[.005 .05]);

% Update the Full Signal axis and create a patch and line object to identify the
% currently previewed portion of the signal
axes(handles.sig);
plot3_handle = plot(time(1:20:end), data(1:20:end,1), 'r-', time(1:20:end),
data(1:20:end,2), 'b-');
axis([0 max(time) -1 1]);
set(handles.sig,'yticklabel','');
xlabel('Time (sec)');
title(['Full Signal']);

x=get(gca,'xlim');
y=get(gca,'ylim');

patch_handle=patch([x(1) x(1)+.5 x(1)+.5 x(1)],[y(1) y(1) y(2) y(2)],[.75 .75 .75]);
```

```
set(patch_handle,'edgecolor',[.9 .9 .9]);
alpha(patch_handle,.6);
plot3_line_handle=line([x(1)+.25 x(1)+.25],[y(1) y(2)]);
set(plot3_line_handle,'color',[1 0 1]);

% Update the GUI handles structure with information that will be needed
% by other callback functions
handles.fname = fname;
handles.fs = fs;
handles.data = data;
handles.time = time;
handles.patch_handle = patch_handle;
handles.plot1_line_handle = plot1_line_handle;
handles.plot2_line_handle = plot2_line_handle;
handles.plot3_line_handle = plot3_line_handle;
handles.plot1_handle = plot1_handle;
handles.plot2_handle = plot2_handle;
handles.plot3_handle = plot3_handle;
handles.plotf_handle = plotf_handle;
handles.plotfL_handle = plotfL_handle;
handles.preview = preview;
guidata(hObject,handles);

% Set the Full Signal axis, plot, and patch to have the same
% buttondownfcn to allow for interactive region selection
set(handles.sig,'ButtonDownFcn',{@selectRegion, handles, 1});
set(handles.plot3_handle,'ButtonDownFcn',{@selectRegion, handles, 1});
set(handles.patch_handle,'ButtonDownFcn','set(findobj("Name","Phase 4:
SoundRecorder"),"UserData","down");');
set(handles.plot3_line_handle,'ButtonDownFcn','set(findobj("Name","Phase 4:
SoundRecorder"),"UserData","down");');
set(handles.figure1,'WindowButtonUpFcn','set(findobj("Name","Phase 4:
SoundRecorder"),"UserData","up");');
set(handles.figure1,'WindowButtonMotionFcn',{@selectRegion, handles, 0});



% --- Hand-written callback function for DAQ object
% --- Executes when the user presses the PLAY button
function play_Callback(hObject, eventdata, handles)
% hObject    handle to play (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Update the handles structure
```

```
handles = guidata(handles.figure1);

data = handles.data;
fs = handles.fs;
plot3_line_handle = handles.plot3_line_handle;

% Use the data that comes after the specified location on the 'Full Signal' axis
pos = get(plot3_line_handle,'XData');
pos = round(pos(1)*fs);
data(1:pos,:) = [];


% Use the analogoutput object to send the data to the soundcard
putdata(handles.daq_object_out, data);
start(handles.daq_object_out);




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  CUSTOM FUNCTIONS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% --- Hand-written callback function for updating the axes
% --- This function controls signal panning
function selectRegion(hObject, eventdata, handles, status)
% hObject    handle to send_to_sptool (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if strcmp(handles.daq_object.running,'On'), return; end

if status == 0
   if strcmp(get(handles.figure1,'UserData'),'up'), return, end
end

% Update the handles structure
handles = guidata(handles.figure1);


try
   state = get(0,'showhiddenhandles');
   set(0,'showhiddenhandles','on');

   patch_handle = handles.patch_handle;
   plot1_line_handle = handles.plot1_line_handle;
   plot2_line_handle = handles.plot2_line_handle;
```

```
plot3_line_handle = handles.plot3_line_handle;
hsig = handles.sig;
data = handles.data;
time = handles.time;
preview = handles.preview;
maxtime = get(handles.chan1slider,'max');
fs = handles.fs;

if status == 2 || strcmp(handles.daq_object_out.running, 'On')
    increment = preview/fs;
    g = get(plot3_line_handle,'XData');
    g = g(2);

    maxtime = maxtime + .25 - increment;
else
    increment = .25;
    g = get(hsig,'CurrentPoint');
    g = g(1);
end

if g > maxtime, g = maxtime; end
if g <= 0, g = 1/fs; end
g = g -.25 + increment;

set(plot3_line_handle,'XData',[g+.25 g+.25]);
set(patch_handle,'XData',[g g+.5 g+.5 g]);

set(handles.chan1slider,'Value',g);

ydata1 = data(ceil(g*fs):ceil(g*fs+fs/2-1),1);
ydata2 = data(ceil(g*fs):ceil(g*fs+fs/2-1),2);
XData = linspace(g, g+.5, length(ydata1));

axes(handles.chan1)
set(handles.plot1_handle,'YData',ydata1);
set(handles.plot1_handle,'XData',XData);
set(plot1_line_handle,'XData',[g+.25 g+.25]);
axis([min(XData) max(XData) -1 1]);

axes(handles.chan2)
set(handles.plot2_handle,'YData',ydata2);
set(handles.plot2_handle,'XData',XData);
set(plot2_line_handle,'XData',[g+.25 g+.25]);
axis([min(XData) max(XData) -1 1]);
```

```
    % Update the frequency axes
    freqstep = (fs/length(ydata1));
    freqvals = 0:freqstep:fs/2;
    Y1 = fft(ydata1);
    Y2 = fft(ydata2);
    Pyy1 = (Y1.* conj(Y1)) / length(ydata1);
    Pyy2 = (Y2.* conj(Y2)) / length(ydata2);
    axes(handles.freq);
    set(handles.plotf_handle,'Xdata',freqvals);
    set(handles.plotf_handle,'Ydata',Pyy1(1:length(ydata1)/2+1));
    axis([freqvals(1) freqvals(length(freqvals)) 0 max(Pyy1(2:length(Pyy1)))]);
    axes(handles.freqL);
    set(handles.plotf_handle,'Xdata',freqvals);
    set(handles.plotfL_handle,'Ydata',Pyy2(1:length(ydata2)/2+1));
    axis([freqvals(1) freqvals(length(freqvals)) 0 max(Pyy2(2:length(Pyy2)))]);



% freqstep = (fs/ceil(fs/2));
% freqvals = 0:freqstep:fs/2;
% Y1 = fft(data(1:ceil(fs/2),1));
%Y2 = fft(data(1:ceil(fs/2),2));
%Pyy1 = (Y1.* conj(Y1)) / ceil(fs/2);
%Pyy2 = (Y2.* conj(Y2)) / ceil(fs/2);
%axes(handles.freq);
%plotf_handle = plot(freqvals,Pyy1(1:ceil(fs/2)/2+1),'r-');
%axis tight;
%set(handles.freq, 'TickLength', [0.01 0]);
%ylabel('Power');
%title('FFT');
%axes(handles.freqL);
%plotfL_handle = plot(freqvals,Pyy2(1:ceil(fs/2)/2+1),'b-');
%axis tight;
%set(handles.freq, 'TickLength', [0.01 0]);
%ylabel('Power');
%xlabel('frequency');
    % axes(handles.freq)
    % set(handles.plotf_handle,'YData',Pyy(1:(preview/2+1),1));
    % set(handles.plotf_handle,'XData',XData);
    % set(plotf_line_handle,'XData',[g+.25 g+.25]);
    % axis([min(XData) max(XData) -1 1]);

    drawnow;

    set(0,'showhiddenhandles',state);
```

```
catch
   errordlg(lasterr,'Error','modal');
   set(0,'showhiddenhandles',state);
end


% --- Hand-written function for starting DAQ object
% --- This function starts the data acquisition session
function status = start_acquisition(handles)
try
   start(handles.daq_object);
   status = 1;
catch
   errordlg(['Error while trying to start acquisition. Please make sure that you are in a
directory with'...
         ' read/write privileges:', sprintf('\n\n') lasterr], 'Error','modal');
   stop(handles.daq_object);
   status = 0;
end


% --- Hand-written function for stoping DAQ object
% --- This function stops the data acquisition session
function status = stop_acquisition(handles)
if isvalid(handles.daq_object)
   stop(handles.daq_object);
   status = 1;
else
   status = 0;
end


% --- Hand-written function for updating plots
% --- This function updates the Left and Right channel plots
function update_plots(plot1_handle, plot2_handle, plotf_handle,plotfL_handle, handles,
preview)
while handles.daq_object.SamplesAcquired < preview, end
while isvalid(handles.daq_object) & strcmp(handles.daq_object.running,'On')
   d = peekdata(handles.daq_object,preview);
   set(plot1_handle,'YData',d(:,1));
   set(plot2_handle,'YData',d(:,2));
   Y = fft(d);
   Pyy = (Y.* conj(Y)) / preview;
   set(plotf_handle,'Ydata',Pyy(1:(preview/2+1),1));
   set(plotfL_handle,'Ydata',Pyy(1:(preview/2+1),2));
```

```
   drawnow;
end



% --- Hand-written callback function for DAQ object
% --- Executed when data acquisition starts
function start_daq(hObject, eventdata, handles)

set(handles.start_stop,'CData',stopbutton,'ToolTipString','Stop sound card
acquisition','Value',1);

% Disable GUI elements that should not be modified during recording
set(handles.fs_text,'Enable','off');
set(handles.samplerate,'Enable','off');
set(handles.dur_text,'Enable','off');
set(handles.duration,'Enable','off');
set(handles.chan1slider,'Enable','off');
set(handles.send_to_workspace,'Enable','off');
set(handles.send_to_sptool,'Enable','off');
set(handles.send_to_figure,'Enable','off');
set(handles.play,'CData',playbutton(1),'Enable','off');
set(handles.save_as,'Enable','off');


% --- Hand-written callback function for DAQ object
% --- Executed when data acquisition stops
function stop_daq(hObject, eventdata, handles)

set(handles.start_stop,'CData',recordbutton,'ToolTipString','Start sound card
acquisition','Value',0);

% Enable GUI elements that can be modified after recording
set(handles.fs_text,'Enable','on');
set(handles.samplerate,'Enable','on');
set(handles.dur_text,'Enable','on');
set(handles.duration,'Enable','on');
set(handles.chan1slider,'Enable','on');
set(handles.play,'CData',playbutton(0),'Enable','on');
set(handles.send_to_workspace,'Enable','on');
set(handles.send_to_sptool,'Enable','on');
set(handles.send_to_figure,'Enable','on');
set(handles.save_as,'Enable','on');
```

```
% --- Hand-written callback function for analogoutput DAQ object
% --- Executed when the analogoutput object starts running
function start_daq_out(hObject, eventdata, handles)
% hObject    handle to daq_object
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.play,'CData',playbutton(1),'Enable','off');
set(handles.start_stop,'Enable','off');


% --- Hand-written callback function for analogoutput DAQ object
% --- Executed when the analogoutput object stops running
function stop_daq_out(hObject, eventdata, handles)
% hObject    handle to daq_object
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

selectRegion(handles.daq_object_out, [], handles, 2);
set(handles.play,'CData',playbutton(0),'Enable','on');
set(handles.start_stop,'Enable','on');


% --- Hand-written callback function for STOP button image
% --- Used to return 'CData' for the Stop icon on the Record\Stop toggle button
function stop = stopbutton

stop = ones(18,18,3);
stop = iconize(imread('stop.jpg'));
%stop(stop == 255) = .8*255;
stop = im2cdata(stop, [255 255 255], 30);


% --- Hand-written callback function for RECORD button image
% --- Used to return 'CData' for the Record icon on the Record\Stop toggle button
function record = recordbutton

record = ones(18,18,3);
record = iconize(imread('record.jpg'));
%record(record == 255) = .8*255;
record = im2cdata(record,[255 255 255], 30);



% --- Hand-written callback function for PLAY button image
```

```
% --- Used to return 'CData' for the Play icon on the Play button
function play = playbutton(status)

play = ones(18,18,3);
if status == 0
   play = iconize(imread('play.jpg'));
else
   play = iconize(imread('play_gray.jpg'));
end
%play(play == 255) = .8*255;
play = im2cdata(play, [255 255 255], 30);


% --- Hand-written callback utility function for button images
% --- Used to create icon data from an image, a
function out = iconize(a)

[r,c,d] = size(a);
r_skip = ceil(r/18);
c_skip = ceil(c/18);

out = a(1:r_skip:end,1:c_skip:end,:);


% --- Executes on mouse press over axes background.
function freq_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to freq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Function A32 – nsom

```
function nsom(time,datapoints,iterations)

piezoport = 'COM1';
serialobject = openpiezocontroller (piezoport);
piezosetlimits (serialobject,0.0,75.0);
piezosetall (serialobject, 1.0);

beginx = 1.0;
beginy = 1.0;
beginz = 1.0;

stepx = 1.0;
stepy = 1.0;
stepz = 0.1;

endx = 1.0;
endy = 1.0;
endz = 50.0;

piezoset (serialobject,'X',beginx);
piezoset (serialobject,'Y',beginy);
piezoset (serialobject,'Z',beginz);

count = 0;

for y = beginy:stepy:endy,
   piezoset (serialobject,'Y',y);

   for x = beginx:stepx:endx,
      piezoset (serialobject,'X',x);

      piezoset (serialobject,'Z',endz);
      pause(0.5);

      for i = 1:1:iterations,
         count = count + 1
         [peakmax(count,:), peaksum(count,:), index(count,:)] =
nsomwavcollect(time,datapoints,1);
         %[peakmax(count,:), peaksum(count,:), index(count,:)] =
nsomwavcollect(time,datapoints,iterations);
      end

      piezoset (serialobject,'Z',beginz);
```

```
        pause(0.5);

    end
end



close all hidden
scrsz = get(0,'ScreenSize');
h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
set(axes,'FontSize',16);
xlabel('Distance (Volts)','FontSize',18);
%xlim([0 (endx)])
ylabel('Signal Intensity','FontSize',18);

hold on

assignin ('base', 'peakmax',peakmax);
assignin ('base', 'peaksum',peaksum);
assignin ('base', 'index',index);

xaxis = 1:1:count;
%xaxis =  beginx:stepx:endx;

plot (xaxis, peaksum, 'r');
plot (xaxis, peakmax, 'g');

closepiezocontroller (serialobject);


end
```

Function A33 – nsom2D

```
function nsom2D1way(time,datapoints,iterations)

% Brackets denote variables that can be set... ie. {time}
%
% This function is designed to operate the NSOM from its current location.
%
% A scan in the "X direction" is completed as follows:
% The NSOM's stage will begin at the home position of {beginx}, {beginy}.
% It will move from {beginx} to {endx} in increments of {stepx}. It will
% then return to beginx and repeat at the next y value incremented by
% {stepy}.  The process with repeat until the final point of {endx}, {endy}
% has been reached. At each position, the NSOM will record {datapoints}
% number of samples from {piezoport} for a duration of {time} in seconds.
%


tic
close all hidden
scrsz = get(0,'ScreenSize');

piezoport = 'COM1';
serialobject = openpiezocontroller (piezoport);
piezosetlimits (serialobject,0.0,75.0);
piezosetall (serialobject, 1.0);

beginx = 1.1;
beginy = 1.1;
beginz = 1.0;

stepx = 0.1;
stepy = 0.1;
stepz = 1.0;

endx = 1.1;
endy = 6.0;
endz = 1.0;

xscans = (((endx-beginx)/stepx)+1);
yscans = (((endy-beginy)/stepy)+1);
totalscans = xscans*yscans;

peakmax = zeros(yscans,xscans);
peaksum = zeros(yscans,xscans);
```

```
index = zeros(yscans,xscans);

for numberofruns = 1:1:1,

   piezoset (serialobject,'X',beginx);
   piezoset (serialobject,'Y',beginy);
   piezoset (serialobject,'Z',beginz);

   count = 0;

         for y = beginy:stepy:endy,
         piezoset (serialobject,'Y',y);

         for x = beginx:stepx:endx,
         piezoset (serialobject,'X',x);

         %piezoset (serialobject,'Z',endz);
         pause(0.05);

         for i = 1:1:iterations,
            count = count + 1;

            fprintf ('\nScan %0.0f of %0.0f\n', count, totalscans)

            [peakmaxtemp, peaksumtemp, indextemp] =
nsomwavcollect(time,datapoints,1);
            peakmax(round(((y-beginy)/stepy)+1),round(((x-beginx)/stepx)+1)) =
peakmaxtemp(1,1);
            peaksum(round(((y-beginy)/stepy)+1),round(((x-beginx)/stepx)+1)) =
peaksumtemp(1,1);
            index(round(((y-beginy)/stepy)+1),round(((x-beginx)/stepx)+1)) =
indextemp(1,1);
            %[peakmax(count,:), peaksum(count,:), index(count,:)] =
nsomwavcollect(time,datapoints,iterations);
         end

         %piezoset (serialobject,'Z',beginz);
         %pause(0.5);
         if numberofruns == 1,
            assignin ('base', 'peakmax1',peakmax);
            assignin ('base', 'peaksum1',peaksum);
            assignin ('base', 'index1',index);
         elseif numberofruns == 2,
            assignin ('base', 'peakmax2',peakmax);
            assignin ('base', 'peaksum2',peaksum);
```

```
            assignin ('base', 'index2',index);
        elseif numberofruns == 3,
            assignin ('base', 'peakmax3',peakmax);
            assignin ('base', 'peaksum3',peaksum);
            assignin ('base', 'index3',index);
        elseif numberofruns == 4,
            assignin ('base', 'peakmax4',peakmax);
            assignin ('base', 'peaksum4',peaksum);
            assignin ('base', 'index4',index);
        end
    end
    elapsedtime = (round(toc/6))/10;
    fprintf('Elapsed time is % 5.1f minutes.\n', elapsedtime)
end

if numberofruns == 1,
    assignin ('base', 'peakmax1',peakmax);
    assignin ('base', 'peaksum1',peaksum);
    assignin ('base', 'index1',index);
elseif numberofruns == 2,
    assignin ('base', 'peakmax2',peakmax);
    assignin ('base', 'peaksum2',peaksum);
    assignin ('base', 'index2',index);
elseif numberofruns == 3,
    assignin ('base', 'peakmax3',peakmax);
    assignin ('base', 'peaksum3',peaksum);
    assignin ('base', 'index3',index);
elseif numberofruns == 4,
    assignin ('base', 'peakmax4',peakmax);
    assignin ('base', 'peaksum4',peaksum);
    assignin ('base', 'index4',index);
end


[y,x] = size(peakmax);
if y > 1 && x > 1,
    x = 0:200*stepx:x*200*stepx-1;
    y = 0:200*stepy:y*200*stepy-1;

    h = figure ('Position', [1 (scrsz(4)-601) 800 600]);

    set(axes,'FontSize',16);
    view(3)
    title ('Scan Results','FontSize',20);
    xlabel ('Distance (nm)','FontSize',18);
```

```
    ylabel ('Distance (nm)','FontSize',18);
    zlabel ('Response');

    mesh(x,y,peakmax,'LineStyle','none','FaceColor','interp');
    camlight left; lighting phong
    %axis vis3d

    grid off
end

end

closepiezocontroller (serialobject);


end
```

Function A34 – nsom2D1way

```
function nsom2D1way(time,datapoints,iterations)

% Brackets denote variables that can be set... ie. {time}
%
% This function is designed to operate the NSOM from its current location.
%
% A scan in the "X direction" is completed as follows:
% The NSOM's stage will begin at the home position of {beginx}, {beginy}.
% It will move from {beginx} to {endx} in increments of {stepx}. It will
% then return to beginx and repeat at the next y value incremented by
% {stepy}.  The process with repeat until the final point of {endx}, {endy}
% has been reached. At each position, the NSOM will record {datapoints}
% number of samples from {piezoport} for a duration of {time} in seconds.
%


tic
close all hidden %closes all of the open graphs
scrsz = get(0,'ScreenSize'); %records your screen resolution for later plotting

piezoport = 'COM1';  %Check your communications port for the piezo controller and edit
here
serialobject = openpiezocontroller (piezoport); %Opens communication with the piezo
controller
piezosetlimits (serialobject,0.0,75.0); %DO NOT CHANGE - This is set for the Melles
Griot Stage
piezosetall (serialobject, 1.0); %This sets the stage to it's initial voltage position

%Edit these parameters for the 2D or 3D scan of your choice
%These values correspond to the Voltage of the piezo controller
%Each Volt Represents 200 nm of movement
%The maximum is 75.0 Volts

beginx = 1.1;
beginy = 1.1;
beginz = 1.0;

stepx = 0.1;
stepy = 0.1;
stepz = 1.0;

endx = 1.1;
endy = 75.0;
```

```
endz = 1.0;

totalruns = 2; %This is the total number of duplicates scanned

%End of editable parameters

xscans = (((endx-beginx)/stepx)+1);
yscans = (((endy-beginy)/stepy)+1);
totalscans = xscans*yscans;

peakmax = zeros(yscans,xscans);
peaksum = zeros(yscans,xscans);
index = zeros(yscans,xscans);

for numberofruns = 1:1:totalruns,

   piezoset (serialobject,'X',beginx);
   piezoset (serialobject,'Y',beginy);
   piezoset (serialobject,'Z',beginz);

   count = 0;

        for y = beginy:stepy:endy,
        piezoset (serialobject,'Y',y);

        for x = beginx:stepx:endx,
        piezoset (serialobject,'X',x);

        %piezoset (serialobject,'Z',endz);
        pause(0.05);

        for i = 1:1:iterations,
           count = count + 1;

           fprintf ('\nScan %0.0f of %0.0f\n', count, totalscans)

           [peakmaxtemp, peaksumtemp, indextemp] =
nsomwavcollect(time,datapoints,1);
           peakmax(round(((y-beginy)/stepy)+1),round(((x-beginx)/stepx)+1)) =
peakmaxtemp(1,1);
           peaksum(round(((y-beginy)/stepy)+1),round(((x-beginx)/stepx)+1)) =
peaksumtemp(1,1);
           index(round(((y-beginy)/stepy)+1),round(((x-beginx)/stepx)+1)) =
indextemp(1,1);
```

```
            %[peakmax(count,:), peaksum(count,:), index(count,:)] =
nsomwavcollect(time,datapoints,iterations);
        end

        %piezoset (serialobject,'Z',beginz);
        %pause(0.5);
        if numberofruns == 1,
            assignin ('base', 'peakmax1',peakmax);
            assignin ('base', 'peaksum1',peaksum);
            assignin ('base', 'index1',index);
        elseif numberofruns == 2,
            assignin ('base', 'peakmax2',peakmax);
            assignin ('base', 'peaksum2',peaksum);
            assignin ('base', 'index2',index);
        elseif numberofruns == 3,
            assignin ('base', 'peakmax3',peakmax);
            assignin ('base', 'peaksum3',peaksum);
            assignin ('base', 'index3',index);
        elseif numberofruns == 4,
            assignin ('base', 'peakmax4',peakmax);
            assignin ('base', 'peaksum4',peaksum);
            assignin ('base', 'index4',index);
        end
    end
    elapsedtime = (round(toc/6))/10;
    fprintf('Elapsed time is % 5.1f minutes.\n', elapsedtime)
end

if numberofruns == 1,
    assignin ('base', 'peakmax1',peakmax);
    assignin ('base', 'peaksum1',peaksum);
    assignin ('base', 'index1',index);
elseif numberofruns == 2,
    assignin ('base', 'peakmax2',peakmax);
    assignin ('base', 'peaksum2',peaksum);
    assignin ('base', 'index2',index);
elseif numberofruns == 3,
    assignin ('base', 'peakmax3',peakmax);
    assignin ('base', 'peaksum3',peaksum);
    assignin ('base', 'index3',index);
elseif numberofruns == 4,
    assignin ('base', 'peakmax4',peakmax);
    assignin ('base', 'peaksum4',peaksum);
    assignin ('base', 'index4',index);
end
```

```
    [y,x] = size(peakmax);
if y > 1 && x > 1,
    x = 0:200*stepx:x*200*stepx-1;
    y = 0:200*stepy:y*200*stepy-1;

    h = figure ('Position', [1 (scrsz(4)-601) 800 600]);

    set(axes,'FontSize',16);
    view(3)
    title ('Scan Results','FontSize',20);
    xlabel ('Distance (nm)','FontSize',18);
    ylabel ('Distance (nm)','FontSize',18);
    zlabel ('Response');

    mesh(x,y,peakmax,'LineStyle','none','FaceColor','interp');
    camlight left; lighting phong
    %axis vis3d

    grid off
end

end

closepiezocontroller (serialobject);


end
```

Function A35 – nsom2D2way

```
function nsom2D2way(time,datapoints,iterations)

% Brackets denote variables that can be set... ie. {time}
%
% This function is designed to operate the NSOM from its current location.
%
% First a scan in the "X direction" is completed as follows:
% The NSOM's stage will begin at the home position of {beginx}, {beginy}.
% It will move from {beginx} to {endx} in increments of {stepx}. It will
% then return to beginx and repeat at the next y value incremented by
% {stepy}.  The process with repeat until the final point of {endx}, {endy}
% has been reached. At each position, the NSOM will record {datapoints}
% number of samples per second from {piezoport} for a duration of {time}
% in seconds.
%
% Next a scan in the "Y direction" is completed as follows:
% The NSOM's stage will again begin from the home position of
% {beginx}, {beginy}. This time, it will move from {beginy} to {endy} in
% increments of {stepy}. It will then return to beginy and repeat at the
% next x value incremented by {stepx}.  The process will repeat until the
% final point of {endx}, {endy} has been reached. At each position, the
% NSOM will record {datapoints} number of samples per second from {piezoport}
% for a duration of {time} in seconds.
%

tic
close all hidden %closes all of the open graphs
scrsz = get(0,'ScreenSize'); %records your screen resolution for later plotting

piezoport = 'COM1';  %Check your communications port for the piezo controller and edit
here
serialobject = openpiezocontroller (piezoport); %Opens communication with the piezo
controller
piezosetlimits (serialobject,0.0,75.0); %DO NOT CHANGE - This is set for the Melles
Griot Stage
piezosetall (serialobject, 1.0); %This sets the stage to it's initial voltage position

%Edit these parameters for the 2D or 3D scan of your choice
%These values correspond to the Voltage of the piezo controller
%Each Volt Represents 200 nm of movement
%The maximum is 75.0 Volts

beginx = 1.1;
```

```
beginy = 1.1;
beginz = 1.1;

stepx = 0.1;
stepy = 0.1;
stepz = 0.1;

endx = 75.0;
endy = 1.1;
endz = 1.1;

totalruns = 1; %This is the total number of duplicates scanned

%End of editable parameters

xscans = (((endx-beginx)/stepx)+1);
yscans = (((endy-beginy)/stepy)+1);
totalscans = xscans*yscans;

peakmax = zeros(yscans,xscans);
peaksum = zeros(yscans,xscans);
index = zeros(yscans,xscans);

for numberofruns = 1:1:totalruns,

   piezoset (serialobject,'X',beginx);
   piezoset (serialobject,'Y',beginy);
   piezoset (serialobject,'Z',beginz);

   count = 0;

   for y = beginy:stepy:endy,
   piezoset (serialobject,'Y',y);

      for x = beginx:stepx:endx,
      piezoset (serialobject,'X',x);

         %piezoset (serialobject,'Z',endz);
         pause(0.05);

         for i = 1:1:iterations,
            count = count + 1;

            fprintf ('\nScan %0.0f of %0.0f\n', count, totalscans)
```

```
            [peakmaxtemp, peaksumtemp, indextemp] =
nsomwavcollect(time,datapoints,1);
            peakmax(round(((y-beginy)/stepy)+1),round(((x-beginx)/stepx)+1)) =
peakmaxtemp(1,1);
            peaksum(round(((y-beginy)/stepy)+1),round(((x-beginx)/stepx)+1)) =
peaksumtemp(1,1);
            index(round(((y-beginy)/stepy)+1),round(((x-beginx)/stepx)+1)) =
indextemp(1,1);
            %[peakmax(count,:), peaksum(count,:), index(count,:)] =
nsomwavcollect(time,datapoints,iterations);
         end

         %piezoset (serialobject,'Z',beginz);
         %pause(0.5);
         if numberofruns == 1,
            assignin ('base', 'peakmax1x',peakmax);
            assignin ('base', 'peaksum1x',peaksum);
            assignin ('base', 'index1x',index);
         elseif numberofruns == 2,
            assignin ('base', 'peakmax2x',peakmax);
            assignin ('base', 'peaksum2x',peaksum);
            assignin ('base', 'index2x',index);
         elseif numberofruns == 3,
            assignin ('base', 'peakmax3x',peakmax);
            assignin ('base', 'peaksum3x',peaksum);
            assignin ('base', 'index3x',index);
         elseif numberofruns == 4,
            assignin ('base', 'peakmax4x',peakmax);
            assignin ('base', 'peaksum4x',peaksum);
            assignin ('base', 'index4x',index);
         end
      end
      elapsedtime = (round(toc/6))/10;
      fprintf('Elapsed time is % 5.1f minutes.\n', elapsedtime)
   end

   if numberofruns == 1,
      assignin ('base', 'peakmax1x',peakmax);
      assignin ('base', 'peaksum1x',peaksum);
      assignin ('base', 'index1x',index);
   elseif numberofruns == 2,
      assignin ('base', 'peakmax2x',peakmax);
      assignin ('base', 'peaksum2x',peaksum);
      assignin ('base', 'index2x',index);
   elseif numberofruns == 3,
```

```
    assignin ('base', 'peakmax3x',peakmax);
    assignin ('base', 'peaksum3x',peaksum);
    assignin ('base', 'index3x',index);
elseif numberofruns == 4,
    assignin ('base', 'peakmax4x',peakmax);
    assignin ('base', 'peaksum4x',peaksum);
    assignin ('base', 'index4x',index);
end

peakmax = zeros(yscans,xscans);
peaksum = zeros(yscans,xscans);
index = zeros(yscans,xscans);

piezoset (serialobject,'X',beginx);
piezoset (serialobject,'Y',beginy);
piezoset (serialobject,'Z',beginz);

count = 0;

for x = beginx:stepx:endx,
piezoset (serialobject,'X',x);

    for y = beginy:stepy:endy,
    piezoset (serialobject,'Y',y);

        %piezoset (serialobject,'Z',endz);
        pause(0.05);

        for i = 1:1:iterations,
            count = count + 1;

            fprintf ('\nScan %0.0f of %0.0f\n', count, totalscans)

            [peakmaxtemp, peaksumtemp, indextemp] =
nsomwavcollect(time,datapoints,1);
            peakmax(round(((y-beginy)/stepy)+1),round(((x-beginx)/stepx)+1)) =
peakmaxtemp(1,1);
            peaksum(round(((y-beginy)/stepy)+1),round(((x-beginx)/stepx)+1)) =
peaksumtemp(1,1);
            index(round(((y-beginy)/stepy)+1),round(((x-beginx)/stepx)+1)) =
indextemp(1,1);
            %[peakmax(count,:), peaksum(count,:), index(count,:)] =
nsomwavcollect(time,datapoints,iterations);
        end
```

```
        %piezoset (serialobject,'Z',beginz);
        %pause(0.5);
        if numberofruns == 1,
           assignin ('base', 'peakmax1y',peakmax);
           assignin ('base', 'peaksum1y',peaksum);
           assignin ('base', 'index1y',index);
        elseif numberofruns == 2,
           assignin ('base', 'peakmax2y',peakmax);
           assignin ('base', 'peaksum2y',peaksum);
           assignin ('base', 'index2y',index);
        elseif numberofruns == 3,
           assignin ('base', 'peakmax3y',peakmax);
           assignin ('base', 'peaksum3y',peaksum);
           assignin ('base', 'index3y',index);
        elseif numberofruns == 4,
           assignin ('base', 'peakmax4y',peakmax);
           assignin ('base', 'peaksum4y',peaksum);
           assignin ('base', 'index4y',index);
        end
     end
     elapsedtime = (round(toc/6))/10;
     fprintf('Elapsed time is % 5.1f minutes.\n', elapsedtime)
  end

  if numberofruns == 1,
     assignin ('base', 'peakmax1y',peakmax);
     assignin ('base', 'peaksum1y',peaksum);
     assignin ('base', 'index1y',index);
  elseif numberofruns == 2,
     assignin ('base', 'peakmax2y',peakmax);
     assignin ('base', 'peaksum2y',peaksum);
     assignin ('base', 'index2y',index);
  elseif numberofruns == 3,
     assignin ('base', 'peakmax3y',peakmax);
     assignin ('base', 'peaksum3y',peaksum);
     assignin ('base', 'index3y',index);
  elseif numberofruns == 4,
     assignin ('base', 'peakmax4y',peakmax);
     assignin ('base', 'peaksum4y',peaksum);
     assignin ('base', 'index4y',index);
  end
end

%     [y,x] = size(peakmax);
% if y > 1 && x > 1,
```

```
%     x = 0:200*stepx:x*200*stepx-1;
%     y = 0:200*stepy:y*200*stepy-1;
%
%     h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
%
%     set(axes,'FontSize',16);
%     view(3)
%     title ('Scan Results','FontSize',20);
%     xlabel ('Distance (nm)','FontSize',18);
%     ylabel ('Distance (nm)','FontSize',18);
%     zlabel ('Response');
%
%     mesh(x,y,peakmax,'LineStyle','none','FaceColor','interp');
%     camlight left; lighting phong
%     %axis vis3d
%
%     grid off
% end



closepiezocontroller (serialobject);


end
```

Function A36 – opencom1

```
function opencom1(s);

s = serial(s);
set(s,'BaudRate',115200);
fopen(s);
s.Timeout = 0.1;
s.Terminator = 'CR/LF';
fprintf(s,'A')
out = fscanf(s)
if out == sprintf ('a\r!');
   fprintf(s,'E');
end
fclose(s)
```

Function A37 – openpiezocontroller

```
function serialobject = openpiezocontroller(port);
%   ie.  openpiezocontroller('COM1')
%   This function initializes the ThorLabs MDT 693A / MDT694A Piezo
%   Contoller.

warning off MATLAB:serial:fscanf:unsuccessfulRead
serialobject = serial(port);
set(serialobject,'BaudRate',115200,'DataBits',8,'Parity','none','StopBits',1,'FlowControl','n
one','Timeout',0.1,'Terminator','CR/LF');
fopen(serialobject);
out = 1;
while out ~= sprintf ('!')
    fprintf(serialobject,'A');
    out = fscanf(serialobject);
    if out(1,1) == sprintf ('!');
        msgbox ('Communication with the Piezo Controller has been Established');
    elseif out(1,1:3) == sprintf ('a\r!');
        fprintf(serialobject,'E');
        out = fscanf(serialobject);
    else
        closepiezocontroller(serialobject);
    end
end
```

Function A38 – piezosetlimits

```
function piezosetlimits (serialport,minvoltage,maxvoltage)

fprintf(serialport,['XL' num2str(minvoltage)]);
out = fscanf(serialport);

fprintf(serialport,['YL' num2str(minvoltage)]);
out = fscanf(serialport);

fprintf(serialport,['ZL' num2str(minvoltage)]);
out = fscanf(serialport);

fprintf(serialport,['XH' num2str(maxvoltage)]);
out = fscanf(serialport);

fprintf(serialport,['YH' num2str(maxvoltage)]);
out = fscanf(serialport);

fprintf(serialport,['ZH' num2str(maxvoltage)]);
out = fscanf(serialport);

fprintf(serialport,'XL?');
XL = fscanf(serialport);

fprintf(serialport,'YL?');
YL = fscanf(serialport);

fprintf(serialport,'ZL?');
ZL = fscanf(serialport);

fprintf(serialport,'XH?');
XH = fscanf(serialport);

fprintf(serialport,'YH?');
YH = fscanf(serialport);

fprintf(serialport,'ZH?');
ZH = fscanf(serialport);

if str2num(XL(1,2)) ~= minvoltage || str2num(YL(1,2)) ~= minvoltage ||
str2num(ZL(1,2)) ~= minvoltage || str2num(XH(1,2:3)) ~= maxvoltage ||
str2num(YH(1,2:3)) ~= maxvoltage || str2num(ZH(1,2:3)) ~= maxvoltage,
   msgbox ('Error Setting All Voltages - Disconnecting Piezo Controller');
   closepiezocontroller (serialport);
```

```
end
```

Function A39 – piezoset

```
function piezoset (serialobject,axis,voltage)

%   axis must be a character input with single quotes
%   ie. piezoset(serialobject,'X',voltage)

if axis ~= 'X' && axis ~= 'Y' && axis ~= 'Z',
   disp ('Please enter an axis: "X", "Y", or "Z"');
else
   fprintf(serialobject,[axis 'V' num2str(voltage)]);
   out = fscanf(serialobject);

%    fprintf(serialobject, [axis 'R?']);
%    axisR = fscanf(serialobject);
%
%    if length(axisR) < 8  || str2num(axisR(1,3:8)) > (voltage+1.5) ||
str2num(axisR(1,3:8)) < (voltage-1.5),
%        out = fscanf(serialobject);
%        out = fscanf(serialobject);
%        clear axisR
%        fprintf(serialobject,[axis 'V' num2str(voltage)]);
%        out = fscanf(serialobject);
%
%        fprintf(serialobject, [axis 'R?']);
%        axisR = fscanf(serialobject);
%
%        if length(axisR) < 8  || str2num(axisR(1,3:8)) > (voltage+1.5) ||
str2num(axisR(1,3:8)) < (voltage-1.5),
%            msgbox (['Error: ' axis '-axis Voltage Could Not Be Set - Disconnecting Piezo
Controller']);
%            closepiezocontroller (serialobject);
%        end
%    else
%        disp ([axis '-axis voltage has been set to ' num2str(voltage) ' volts.'])
%    end
end
```

Function A40 – piezosetall

```
function piezosetall (serialobject,voltage)

fprintf(serialobject,['AV' num2str(voltage)]);
out = fscanf(serialobject);

fprintf(serialobject,'XR?');
XR = fscanf(serialobject);

fprintf(serialobject,'YR?');
YR = fscanf(serialobject);

fprintf(serialobject,'ZR?');
ZR = fscanf(serialobject);

if length(XR) < 8 || length(YR) < 8 || length(ZR) < 8 || str2num(XR(1,3:8)) >
(voltage+1.5) || str2num(XR(1,3:8)) < (voltage-1.5) || str2num(YR(1,3:8)) >
(voltage+1.5) || str2num(YR(1,3:8)) < (voltage-1.5) || str2num(ZR(1,3:8)) > (voltage+1.5)
|| str2num(ZR(1,3:8)) < (voltage-1.5),
   msgbox ('Error: All Voltages Could Not Be Set - Disconnecting Piezo Controller');
   closepiezocontroller (serialobject);
else
   disp (['Voltage has been set to ' num2str(voltage) ' volts.'])
end
```

Function A41 – piezotrial

```
function piezotrial

serialport = 'COM1';
serialport = openpiezocontroller (serialport);
piezosetlimits (serialport,0.0,75.0);
% pause
% piezosetall (serialport,25.0);
% pause
% piezosetall (serialport,50.0);
% pause
% piezosetall (serialport,75.0);
% pause
% piezosetall (serialport,49.5);
% pause
% piezosetall (serialport,24.5);
% pause
closepiezocontroller (serialport);

end
```

Function A42 – nsomwavcollect

```
function [peakmax, peaksum, index] = nsomwavcollect(time,datapoints,iterations)

% time = the total collection time
% datapoints = the number of points collected per second (Frequency of data
%          collection)
% iterations = the number of times the collection process is completed

out = zeros(2,time*datapoints);
peakmax = zeros(iterations,2);
peaksum = zeros(iterations,2);
index = zeros(iterations,2);


for i = 1:1:iterations,
   out = (wavrecord(time*datapoints,datapoints,2))';
   [peakmax(i,1), peaksum(i,1), index(i,1)] = soundprocess(out(1,:),time,datapoints,1);
   [peakmax(i,2), peaksum(i,2), index(i,2)] = soundprocess(out(2,:),time,datapoints,1);
%    fprintf ('Scan %0.0f of %0.0f\n\n', i, iterations)
%    fprintf ('Peak Maximum 1 = %0.4e\n', peakmax(i,1))
%    fprintf ('Peak Maximum 2 = %0.4e\n\n', peakmax(i,2))
%    fprintf ('Peak Integration 1 = %0.4e\n', peaksum(i,1))
%    fprintf ('Peak Integration 2 = %0.4e\n\n', peaksum(i,2))
    fprintf ('Index 1 = %0.0f\n', index(i,1))
%    fprintf ('Index 2 = %0.0f\n\n\n', index(i,2))

end
```

Function A43 – soundprocess

```
function [peakmax,peaksum,index] = soundprocess(sample1,time,datapoints,iterations)

numpoints = (time*datapoints);

fftline = zeros(iterations,numpoints);
cfftline = zeros(iterations,numpoints);
xaxis = time:time:time*iterations;

wind = window(@flattopwin,time*datapoints)';% Change the type of window from the
examples below

%15.2    @bartlett      - Bartlett window.
%15.8    @barthannwin   - Modified Bartlett-Hanning window.
%11.5    @blackman      - Blackman window.
%8.6     @blackmanharris - Minimum 4-term Blackman-Harris window.
%10.7    @bohmanwin     - Bohman window.
%9.1     @chebwin       - Chebyshev window.
%3.7     @flattopwin    - Flat Top window.
%15.4    @gausswin      - Gaussian window.
%18.0    @hamming       - Hamming window.
%15.9    @hann          - Hann window.
%62.9    @kaiser        - Kaiser window.
%8.7     @nuttallwin    - Nuttall defined minimum 4-term Blackman-Harris window.
%9.3     @parzenwin     - Parzen (de la Valle-Poussin) window.
%66.0    @rectwin       - Rectangular window.
%34.1    @tukeywin      - Tukey window.
%15.2    @triang        - Triangular window.

for i = 1:1:iterations,
   fftline(i,:) = fft(sample1(i,:).*wind);
   holder = (fftline(i,:).*conj(fftline(i,:)))/(length(sample1(i,:)));
   cfftline(i,:) = holder(1,1:numpoints);
end

[peakmax, index] = max(cfftline(:,10:(numpoints/2))'); %peak maximum

index = index + 10;

peaksum = (sum(cfftline(:,(index-10):(index+10))')); %integration under the peak
```

Function A44 – nsomsamplerelocate

```
function nsomsamplerelocate(cycles)

piezoport = 'COM1';
serialobject = openpiezocontroller (piezoport);
piezosetlimits (serialobject,0.0,75.0);
piezosetall (serialobject, 1.0);

beginx = 1.0;
beginy = 1.0;
beginz = 1.0;

stepx = 1.0;
stepy = 1.0;
stepz = 0.1;

endx = 75.0;
endy = 1.0;
endz = 50.0;

for i = 1:1:cycles,

   piezoset (serialobject,'X',endx);
   piezoset (serialobject,'Y',endy);
   piezoset (serialobject,'Z',endz);

   count = 0;

   for y = endy:-stepy:beginy,
      piezoset (serialobject,'Y',y);

      for x = endx:-stepx*25:beginx,
         piezoset (serialobject,'X',x);

%           piezoset (serialobject,'Z',endnz);
%           pause(0.5);
%
%           piezoset (serialobject,'Z',beginz);
%           pause(0.5);

      end
   end
   piezoset (serialobject,'Z',beginz);
end
```

```
closepiezocontroller (serialobject);
```

Function A45 – nsomtriplicate

```
function nsomtriplicate(time,datapoints,iterations)

tic
close all hidden
scrsz = get(0,'ScreenSize');

piezoport = 'COM1';
serialobject = openpiezocontroller (piezoport);
piezosetlimits (serialobject,0.0,75.0);
piezosetall (serialobject, 1.0);

beginx = 1.0;
beginy = 1.0;
beginz = 25.0;

stepx = 1.0;
stepy = 1.0;
stepz = 1.0;

endx = 75.0;
endy = 75.0;
endz = 25.0;

totalscans = ((endy-beginy+1)/stepy)*((endx-beginx+1)/stepx);

for numberofruns = 1:1:1,

   piezoset (serialobject,'X',beginx);
   piezoset (serialobject,'Y',beginy);
   piezoset (serialobject,'Z',beginz);

   count = 0;


   for y = beginy:stepy:endy,
      piezoset (serialobject,'Y',y);
      for x = beginx:stepx:endx,
         piezoset (serialobject,'X',x);

         %piezoset (serialobject,'Z',endz);
         pause(0.05);

         for i = 1:1:iterations,
```

```
            count = count + 1;
            fprintf ('Scan %0.0f of %0.0f\n\n', count, totalscans)

            [peakmax(count,:), peaksum(count,:), index(count,:)] =
nsomwavcollect(time,datapoints,1);
            %[peakmax(count,:), peaksum(count,:), index(count,:)] =
nsomwavcollect(time,datapoints,iterations);
         end

         %piezoset (serialobject,'Z',beginz);
         %pause(0.5);
         if numberofruns == 1,
            assignin ('base', 'peakmax1',peakmax);
            assignin ('base', 'peaksum1',peaksum);
            assignin ('base', 'index1',index);
         elseif numberofruns == 2,
            assignin ('base', 'peakmax2',peakmax);
            assignin ('base', 'peaksum2',peaksum);
            assignin ('base', 'index2',index);
         elseif numberofruns == 3,
            assignin ('base', 'peakmax3',peakmax);
            assignin ('base', 'peaksum3',peaksum);
            assignin ('base', 'index3',index);
         end
      end
      elapsedtime = (round(toc/6))/10;
      fprintf('Elapsed time is % 5.1f minutes.\n', elapsedtime)
   end

   if numberofruns == 1,
      assignin ('base', 'peakmax1',peakmax);
      assignin ('base', 'peaksum1',peaksum);
      assignin ('base', 'index1',index);
   elseif numberofruns == 2,
      assignin ('base', 'peakmax2',peakmax);
      assignin ('base', 'peaksum2',peaksum);
      assignin ('base', 'index2',index);
   elseif numberofruns == 3,
      assignin ('base', 'peakmax3',peakmax);
      assignin ('base', 'peaksum3',peaksum);
      assignin ('base', 'index3',index);
   end

   h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
   set(axes,'FontSize',16);
```

```
    xlabel('Distance (Volts)','FontSize',18);
    %xlim([0 (endx)])
    ylabel('Signal Intensity','FontSize',18);

    hold on

    xaxis = 1:1:count;
    %xaxis =  beginx:stepx:endx;

    plot (xaxis, peaksum, 'r');
    plot (xaxis, peakmax, 'g');
    hold off
end




closepiezocontroller (serialobject);


end
```

Function A46 – plotnsomsample

```
function plotnsomsample(peakmax,peaksum)

    close all hidden
    scrsz = get(0,'ScreenSize');
    h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
    set(axes,'FontSize',16);
    xlabel('Distance (nm)','FontSize',18);
    %xlim([0 (endx)])
    ylabel('Signal Intensity','FontSize',18);
    hold on
    xaxis = 0:1:74;
    %xaxis =  beginx:stepx:endx;
    plot (xaxis*50, peakmax, 'g');
    plot (xaxis*50, peaksum, 'r');
```

Function A47 – closepiezocontroller

```
function closepiezocontroller(serialobject)
try
   fprintf(serialobject,['AV' num2str(0.5)]);
   out = fscanf(serialobject);
   fclose(serialobject);
   msgbox ('The Piezo Controller has been Disconnected');
catch
   msgbox('Communications with the Piezo Controller could not be Disconnected.
Please close MatLab and try again.');
end
```

Function A48 – elementsequal

```
function out = elementsequal(datain1,datain2)

if size(datain1) == size(datain2)

   [x,y] = size(datain1);

   for i = 1:1:x,
      for j = 1:1:y,
         if datain1(i,j) == datain2(i,j),
            out(i,j)=1;
         else
            out(i,j)=0;
         end
      end
   end
end

end
```

Function A49 – reducemaxheights

```
function datain = reducemaxheights(datain, maxheight, newheight)

[i,j] = size(datain);

for i = 1:1:i,
   for j = 1:1:j,
      if datain(i,j) > maxheight,
         datain(i,j) = newheight;
      end
   end
end
```

Function A50 – nsomreshape

```
function reshaped = nsomreshape(data,rows,columns)
for i = 1:1:rows,
   reshaped (i,:) = data((((i-1)*columns)+1):(i*columns),1);
end
```

Function A51 – nsom2dplot

```
function nsom2dplot(data,stepsize,name,peaksum)

% data = the data output from MatLab to the workspace
% stepsize = the stepsize in Volts

% nsom2dplot plots several different 2-D surface plots of different types
% The plots are titles based upon the plot type and the x and y axis are
% labeled in nanometers.  It is assumed that that stepsize is the same in
% both the x and y directions.  If this is not so, stepsize may be changed
% below for the appropriate stepsize in each direction.

close all hidden

[y,x] = size(data);

if x > 1 && y > 1,
x = 0:200*stepsize:x*200*stepsize-1;
y = 0:200*stepsize:y*200*stepsize-1;

figure
set(axes,'FontSize',14);
hold on
mesh(x,y,data,'LineStyle','none','FaceColor','interp');
camlight left; lighting phong
xlabel ('Distance (nm) xaxis','FontSize',18);
ylabel ('Distance (nm) yaxis','FontSize',18);
zlabel ('Response','FontSize',18);
title ('Surf','FontSize',20);
%axis vis3d
view(3)
grid off
hold off

figure
set(axes,'FontSize',14);
hold on
surf(x,y,data,'LineStyle','none','FaceColor','interp');
camlight right; lighting phong
xlabel ('Distance (nm) xaxis','FontSize',18);
ylabel ('Distance (nm) yaxis','FontSize',18);
zlabel ('Response','FontSize',18);
title ('Surf','FontSize',20);
%axis vis3d
```

```
view(3)
grid off
hold off

figure
set(axes,'FontSize',14);
hold on
surfl(x,y,data);
camlight left; lighting phong
xlabel ('Distance (nm) xaxis','FontSize',18);
ylabel ('Distance (nm) yaxis','FontSize',18);
zlabel ('Response','FontSize',18);
title ('Surfl','FontSize',20);
%axis vis3d
shading interp
view(3)
grid off
hold off

figure
set(axes,'FontSize',14);
hold on
surfc(x,y,data,'LineStyle','none','FaceColor','interp');
camlight left; lighting phong
xlabel ('Distance (nm) xaxis','FontSize',18);
ylabel ('Distance (nm) yaxis','FontSize',18);
zlabel ('Response','FontSize',18);
title ('Surfc','FontSize',20);
% axis vis3d
view(3)
grid off
hold off

scrsz = get(0,'ScreenSize');
h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
set(axes,'FontSize',14);
hold on
set(gca,'Ygrid','on');
[C,h] = contourf(x,y,data,'LineStyle','none');
xlabel ('Distance (nm) xaxis','FontSize',18);
ylabel ('Distance (nm) yaxis','FontSize',18);
title (strcat([name]),'FontSize',20);
LevelStep = get(h,'LevelStep');
%set(h,'LevelStep',LevelStep*1.25);
%LevelList = get(h,'LevelList');
```

```
%set(h,'LevelList',LevelList(1,11:17));
set(gca,'Layer','top');
get(gca,'YTick');
%set(gca,'YTick',[0 500 1000 1500 2000 2500 3000 3500]);
%set(gca,'GridLineStyle','-');
%set(gca,'TickDir','out')
%set(gca,'YMinorTick','on');
axis tight
colormap jet
hold off
saveas (h,strcat([name '.bmp']));

figure
set(axes,'FontSize',14);
hold on
xlabel ('Distance (nm) xaxis','FontSize',18);
ylabel ('Distance (nm) yaxis','FontSize',18);
zlabel ('Response','FontSize',18);
title ('Imagesc','FontSize',20);
colormap(gray);
imagesc (x,y,data)
%grid off
hold off

else,
   x = 0:200*stepsize:x*200*stepsize-1;
   y = 0:200*stepsize:y*200*stepsize-1;

   peakmax = data;
   scrsz = get(0,'ScreenSize');
   h = figure ('Position', [1 (scrsz(4)-601) 800 600]);
   set(axes,'FontSize',14);
   hold on
   xlabel('Distance (nm)','FontSize',18);
   %xlim([0 (endx)])
   ylabel('Signal Intensity','FontSize',18);
   title (strcat([name]),'FontSize',20);
   %xaxis = 0:1:74;
   %xaxis =  beginx:stepx:endx;
   plot (y, peakmax, 'g');
   plot (y, peaksum, 'r');
   hold off
   saveas (h,strcat([name '.bmp']));
end
```

Appendix B – SSSI Manual

Contents

Software Installation

Install MatLab 7.0.1

Install MatLab files needed for running the SSSI (Zipped SSSI Folder - Unzip into MatLab work folder)

Hardware Setup

The SSSI has an on board 12 VDC gel lead acid battery as its power supply. A 10 VDC voltage regulator is used to regulate the voltage to the controller and diodes. After each use, the battery must be recharged with the charger. Care must be taken to not plug the charger into the 10 VDC regulated supply. Although the 12 VDC will not harm the controller, the diodes will burn out at the higher voltage. Charging of the battery is complete when the charger light switches from red (charging) to green (charging complete). When running the SSSI, the charger may be plugged in but results may be erroneous.

There is a simple power switch for powering the SSSI on and off (Figure B.1).

An NPN phototransistor is used as the detector for the SSSI. It is attached to the SSSI via 1/8 inch stereo phone plug. This detector may be plugged in or removed at any time.

The serial connection may be made via USB-serial adaptor or serial-serial from the computer. The serial connection may also be plugged in or removed at any time.

The battery may be replaced at any time without the controller losing its programming. The supply from the battery is polarity protected.

SSSI GUI Initialization

To open the SSSI GUI, MatLab 7.0.1 is used. Open MatLab and at the command Line type guide and press enter. A GUIDE Quick Start Window will appear after a few seconds (Figure B.2).

Click on the Open Existing GUI tab (Figure B.3).

In this image, the SSSIgui.fig is already showing. The first time SSSIgui.fig is opened you will need to click Browse and navigate to the SSSIgui.fig file which is in your MatLab work folder under SSSI. You placed it there when unzipping the SSSI software at the beginning of this document. It will show up as SSSIgui.fig in the recently opened files list the next time you open GUIDE.

Once you have found the file, open it. A small box will appear stating that GUIDE is initializing (Figure B.4).

Next, the SSSIgui.fig figure will open; Click the play button that is circled in Figure B.5. Another window with the SSSIgui will open (Figure B.6).

Scanning with the SSSI

Connect your serial-serial or USB-serial to the SSSI.

Turn on the power switch to the SSSI.

The Port must be selected using the drop-down menu in the SSSIgui. If it is not correctly selected, the SSSI will not scan.

The connection with the SSSI is made by pressing the Open SSSI button.

The SSSI diode head needs to be placed near the target with the detector able to pick up the reflected or transmitted light.

Now everything is ready to begin scanning using the SSSI.

Pressing the Single Scan button will take a single scan and the results will be shown in the form of a bar graph in the SSSIgui window. The scale is automatically adjusted.

Pressing the 100 Scans button will scan 100 times and then stop.

To save the data while scanning, the Start Data File button needs to be pressed and a directory and filename needs to be determined for the Save to TXT File dialog box. When the scans are completed, the Close Data File button must be pressed.

When scanning is complete, the Close SSSI button must be pressed before closing the SSSIgui, SSSIgui.fig window, and MatLab. The text file will be stored in the selected location.

Troubleshooting

A diode will not light:

- Check resistors in the SSSI housing to be certain that none are loose or touching the silver transistor tops.
- Check the diode itself to be certain that it is not loose in its socket
- Check another diode in the socket of the first to be certain that the first diode is not burnt out

The SSSIgui freezes:

- Try the Close SSSI button
- Try to close the SSSIgui
- Try to close MatLab
- If none of the above work press Ctrl-Alt-Del, select MATLAB, and click End Task.

Appendix B Figures

Figure B.1: Connection end of the SSSI enclosure.

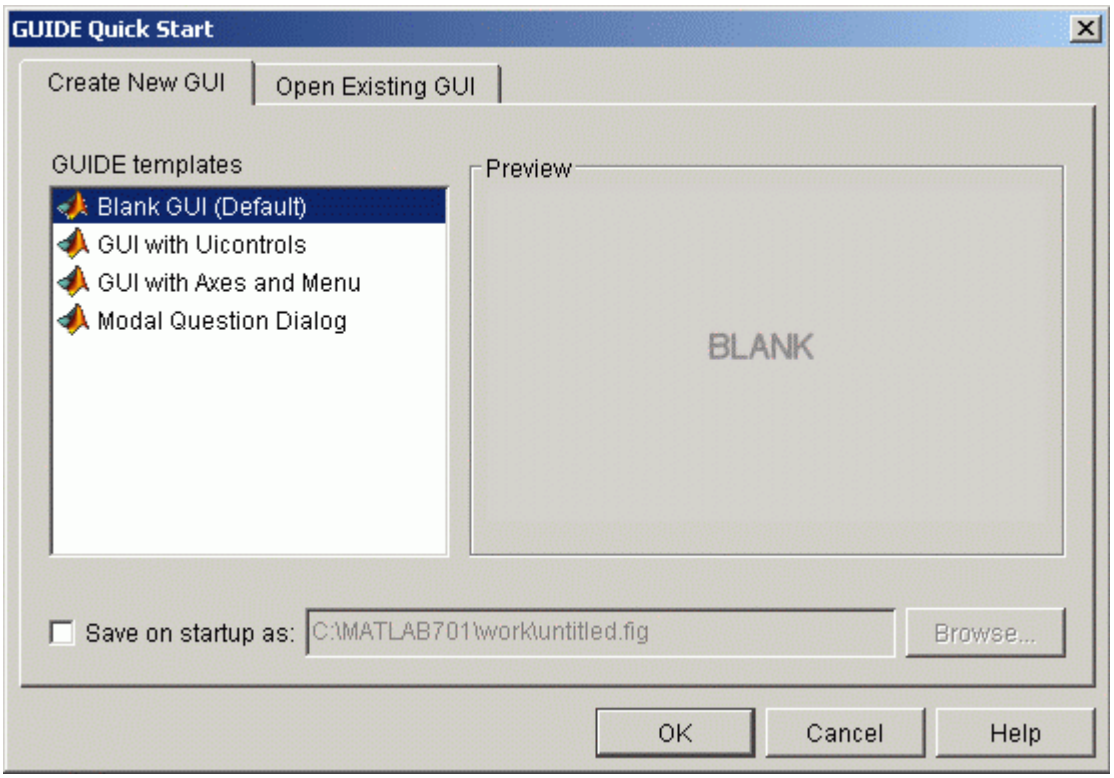

Figure B.2: GUIDE Quick Start window in MatLab.

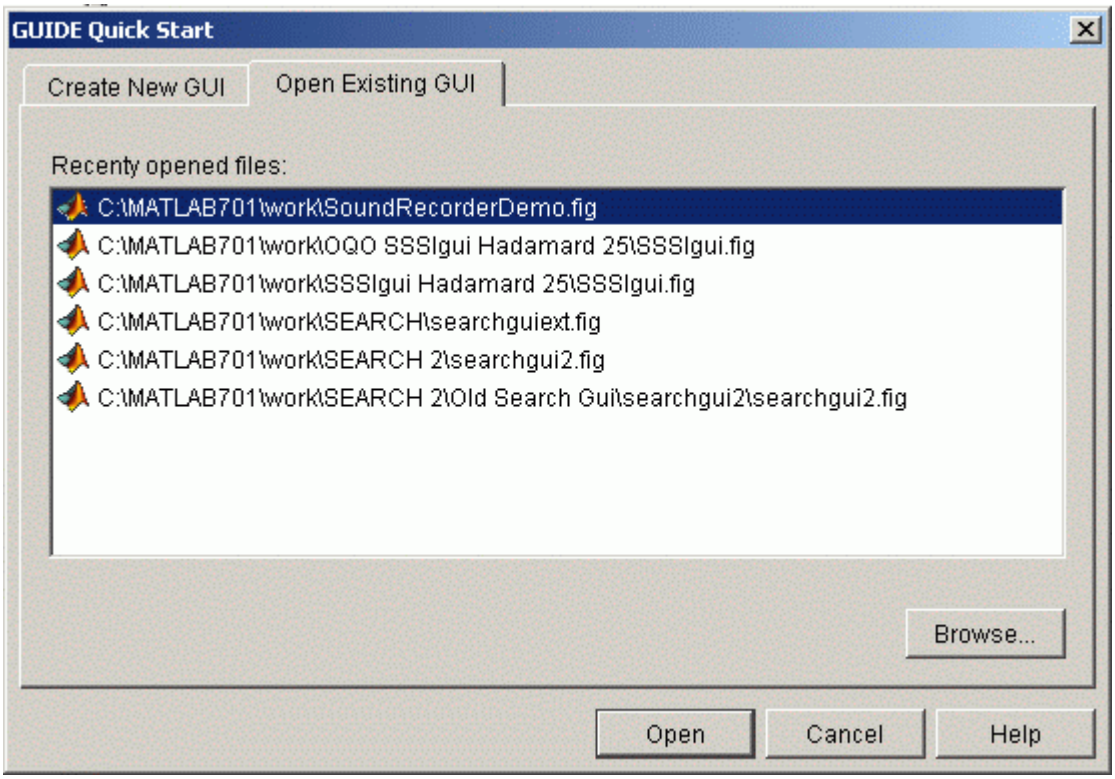

Figure B.3: GUIDE Quick Start window in MatLab with the Open Existing GUI tab selected.

Figure B.4: GUIDE initialization window.

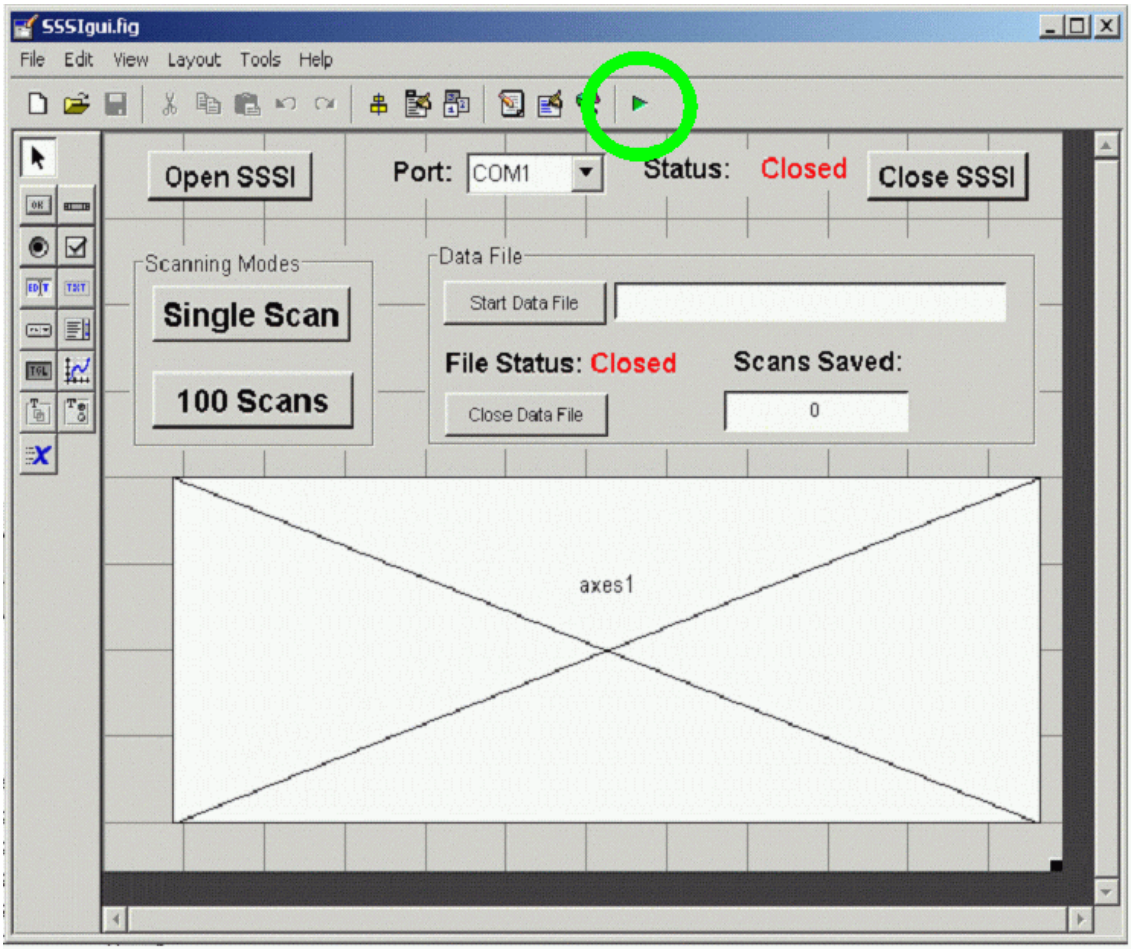

Figure B.5: SSSIgui.fig window with the play button circled in green.

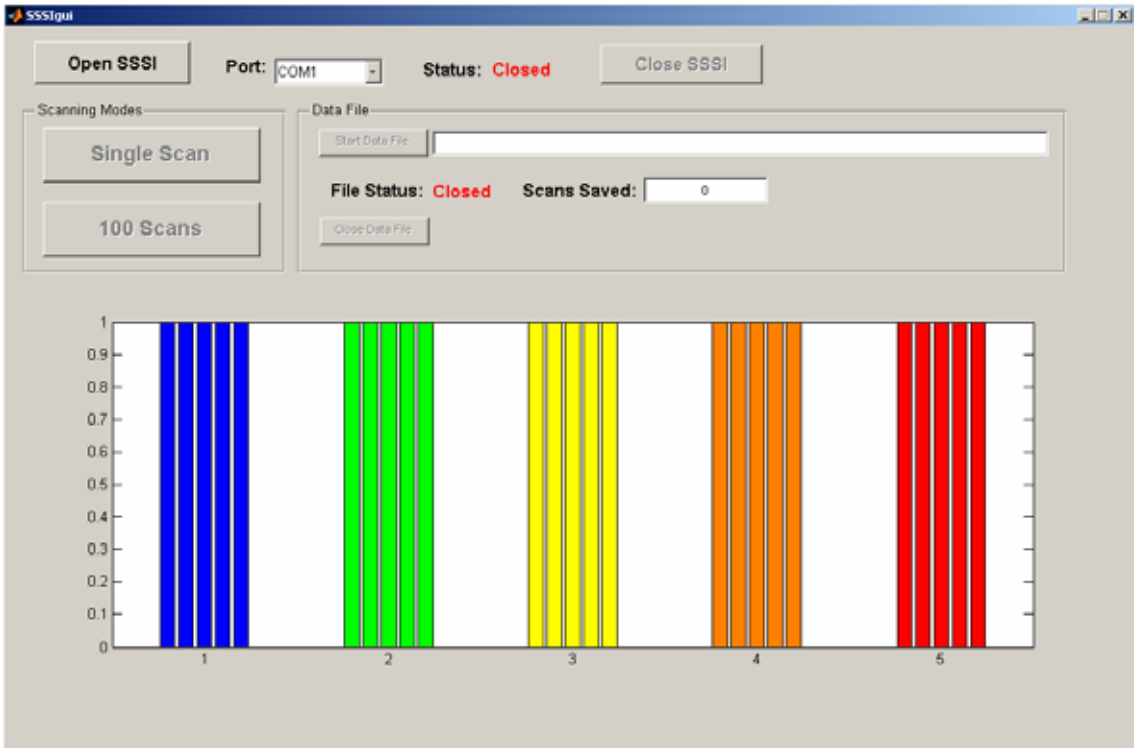

Figure B.6: SSSIgui window.

Appendix C – NSOM Manual

This manual was designed for the technical user that has background knowledge in the theory behind NSOM. This manual will guide you through setting up and connecting the different parts of the NSOM as well as scanning samples in transmission mode.

Contents

Software Needed

MatLab 7.0.1

WinRadio (WR-1550) (Software and Manual)

Creative Sound Blaster MP3+ external sound card (Drivers)

MatLab files needed for running the NSOM (Zipped NSOM Folder - Unzip into MatLab work folder)

Connecting the NSOM box to the beamline

The incoming beam line is ISO-80 standard and connects to the NSOM box with eight 2" x M8 bolts. The bolts pass through the beamline flange (or rotating ring flange), then through our rotatable ring flange with the optical lens (ISO-80 IR optical glass viewport for the end of the beamline - order sheet - flange detail - viewport details) and high vacuum seal, through the cutout ring designed for the manual shutter, and finally the bolts thread directly into the NSOM box (no nuts are required) (Figure C.1).

Setting up the internal components of the NSOM

In the light coupling side of the NSOM box (the larger half), there is a 12" focusing lens (Thor labs lens LB1917 dia = 50.8 mm efl = 300.0 mm), and 3-D micro stage, the fiberoptic chuck (Back-Loading Collet Style Fiber Chuck (Thor Labs HFC005)), and the Pitch & Yaw Fiber Chuck Mount. The layout is shown in the image below. The fiberoptic is fed through the fiber chuck and then inserted into the fiber chuck mount (frequently easier to simply tape the fiber to the chuck). The incoming beam is coupled at the focus point of the light by moving the 3D stage, fiber chuck, and the chuck mount. Correct alignment including pitch and yaw is required for the best signal (Figures C.2-C.3).

NOTE: The pitch and yaw of the incoming light makes a significant (not yet quantified) impact on the amount of NIR light coupled into a 5 μm core fiber.

On the detection side of the NSOM box, we have a 3-D stage mounted on an optical rod for holding the pulled end of the fiberoptic , the Melles Griot Nanometer Stage, the InGaAs Photodiode detector, the WinRadio, and the SoundBlaster sound card. The fiber is attached to the 3-D stage by taping it with cellulose tape (Scotch tape) to a rubber stopper (to help dampen vibrations), which is in turn mounted on the stage. The height and position of the fiber tip may be adjusted with the 3-D stage and there are also coarse adjustments for all three axis on the Nanometer Stage (Figures C.4-C.5). Gentle tapping of either stage induces vibrations in the tip of the fiber probe. Dampening of these vibrations may be visibly noticed under 80X microscope magnification and indicate close proximity of the tip to the sample.

Connecting the electronics of the NSOM

Melles Griot Nanometer Stage and ThorLabs Piezo Controller.

In this section we will cover connections for nanostage movement, radio receiver control, and signal transmission.

NOTE: All cables MUST be thoroughly shielded from intense RF noise (the synchrotron).

The nanometer stage movement is controlled with a ThorLabs Piezo Controller. The Piezo Controller is current limited and provides potentials up to 150 V. The potential across the piezos in our Melles Griot nanometer stage must not exceed 75 V. This potential may be limited both by a switch on the back of the piezo controller and in the

software. The piezo controller is able to precisely control voltages to 0.1 V. This is equivalent to an approximate 20 nm step.

NOTE: Thermal movement with the Melles Griot stage is 1000 nm/° C.

The Piezo Controller requires a serial connection to a computer. This may be connected via standard RS232 9-pin serial cable, or with a USB to serial adapter. See the Piezo Controller Manual for details. Communication with the controller is established directly from MatLab through its COM port. **NOTE**: It is important for communication to verify your communications port connection (COM number). This must be set in the MatLab software (Details further down).

The nanometer stage is connected to the Piezo Controller via three 75 V SMC transfer lines. BNC to SMC adapters are required at the back of the piezo controller. **NOTE:** The SMC transfer lines are extremely sensitive to external noise. These cables and connections must be shielded and grounded carefully.

Power is provided to the nanostage via the piezo controller, which is powered from a 120 VAC wall outlet. Currently, rough adjustments must be completed by hand. We do not have a stepper controller capable of working with the steppers currently mounted on the nanostage.

InGaAs Photodiode Detector, WinRadio, and SoundBlaster External Sound Card.

In this section, we will cover connections for the InGaAs Photodiode Detector, the WinRadio (WR-1550) (Software and Manual), the Creative Sound Blaster MP3+ external sound card (Drivers), and general signal transmission.

The InGaAs photodiode is connected to the WinRadio via a 1' BNC cable connector. The WinRadio is connected to a computer via a standard RS232 9-pin serial cable or with a USB to serial adapter. The WinRadio may be disconnected from the computer's serial port after it is configured. The WinRadio is powered by a 12 VDC source (lawnmower battery). See the WinRadio manual for more information. An 1/8" stereo audio patch cable connects the WinRadio to the Sound Blaster external sound card. Although the sound card may accept the 1/8" stereo connector, it has been found that the RCA inputs on the sound card are less noisy. So, a 1/8" female stereo to two RCA male adapter may be employed if preferred. The sound card is connected to and powered by a USB cable to the computer's USB port.

NOTE: All cables are extremely sensitive to external noise. These cables and connections must be shielded with aluminum foil and grounded carefully (Figure C.6).

Setting up communications with NSOM hardware

Now that all of the physical connections are made, we can cover how to communicate with all of the hardware. The drivers and software linked in the previous section for the WinRadio and Sound Card should be installed if they have not been already.

WinRadio.

The WinRadio is controlled via its own software and communications are initialized upon opening its software. Please be sure the WinRadio is plugged in and powered on before the software is opened. The frequency of the synchrotron is around 52.887 MHz. For data collection, the Lower Side Band Mode (LSB) is chosen and then tuning above 52.887 MHz by 1000 Hz to 52.987 MHz allows an output of 1 KHz and is easily digitized by the sound card. Choose DX (primarily used and needed) for increased sensitivity or Local for lowered sensitivity (Figure C.7). The Squelch and IF Shift should be set to 0 and no scanning will be necessary. The Gain is adjusted by adjusting the Volume level.

NOTE: After the WinRadio is set up, the program window may be closed and the serial connector disconnected. The WinRadio will continue operating at its programmed settings with no problems until the power source is interrupted.

Creative Sound Blaster Sound Card.

After the drivers are installed, communication with the Sound Card is completed automatically when it is plugged into a powered USB port. Recording will be controlled with MatLab, however the Windows Sounds and Audio Devices properties must be configured correctly. To do this, go to Start, Control Panel, and then Sounds and Audio Devices (Figure C.8).

On the volume tab under Device Volume, click Advanced (Figure C.9).

The Volume Control counsel will be displayed (Figure C.10).

NOTE: Alternatively, you may simply double click the speaker icon down by the clock (Figure C.11).

Choose Options and make sure Advanced Controls is checkmarked (if not click it). Then again choose Options and choose Properties (Figure C.12).

This will display the Audio Properties window with the Playback radio button selected. Select the Recording radio button and then be sure Line In and Microphone are the only two boxes checkmarked (Figure C.13).

Click OK and the Recording Control window will be opened (Figure C.14). In this window, we must choose (with the check box) whether we are using the Line In with the 1/8" stereo to RCA adapter or going straight in the Microphone port with the 1/8" stereo patch cable. The most useful gain adjustment is accomplished using the Volume controls in the Recording Control window.

Melles Griot Nanometer Stage and ThorLabs Piezo Controller.

The Melles Griot Nanometer Stage is completely powered and controlled with the ThorLabs Piezo Controller. All communication with the Piezo Controller is maintained through MatLab. The only setting that is required is selection of the appropriate COM port number in the nsom2D2way.m file (Figure C.15). This is listed as the variable

piezoport within the .m-file. If you are unsure of the correct COM port, it may be found by going to your Device Manager (Start, Control Panel, System, Hardware Tab) and looking at Ports.

Adjusting the incoming gain and checking the noise at the sound card

Windows sound recorder only allows sound signals between 0 and 1 V, so care must be taken to not clip the incoming signal. This section describes how to make certain that the collected signal never exceeds the 1 V maximum and thus never getting clipped. In addition, all of the wires used to connect to the electronics within the aluminum NSOM box act as antennas. With the giant RF emitter only feet away (the synchrotron), these wires easily pick up emission at exactly the frequency at which collection occurs. This section will deal with observing and handling noise problems.

Adjusting Gain.

To ensure the maximum light signal is reaching the detector, either no sample or the least absorbing part of the sample should be chosen for gain adjustment. Also, the beam shutter must be open and the fiber aligned to the incoming beam.

To view the incoming signal, the SoundRecorder in MatLab 7.0.1 is used. Open MatLab and at the command Line type guide and press enter. A GUIDE Quick Start Window will appear after a few seconds (Figure C.16).

Click on the Open Existing GUI tab (Figure C.17).

In this image, the SoundRecorderDemo.fig is already showing (SoundRecorder.fig is what is used). The first time SoundRecorder.fig is opened you will need to click Browse and navigate to the SoundRecorder.fig file which is in your MatLab work folder under NSOM. You placed it there when unzipping the NSOM software at the beginning of this document. It will show up as SoundRecorder.fig in the Recently opened files list the next time you open GUIDE.

Once you have found the file, open it. A small box will appear stating that GUIDE is initializing (Figure C.18).

Next, the SoundRecorderDemo.fig figure will open; Click the play button that is circled in Figure C.19.

Another window with the soundrecorder gui will open (Figure C.20). You will need to set a Sample rate and Duration indicated by the Green Arrows below. A sample rate of 8000 for 30 seconds is suggested as a starting point. Now press the record button at the bottom; it is indicated by the red arrow.

After pressing the record button, you will see your signal in the form of an interferogram in the current signal box. The left and right channels are displayed independently. Assuming that everything has been set up correctly a signal will be present. The FFT of the interferogram will be displayed and updated every 0.1 seconds for the left and right channels.

MatLab's sound recording through Windows clips all signals that are above 1.0 V. It is therefore necessary to decrease gain until all possible signals are less than 1 V. So, now you must use the SoundRecorder to view the incoming signal and decrease the gain using the recording volume controls and the WinRadio volume controls until all signals will be less than 1.0 V. 1.0 V is the maximum and minimum viewable in the current signal display areas.

Checking Noise.

Once the gain is set, the amount of noise may be determined by closing the shutter and visually observing the FFT peak height/area. From past experience, signals are on the order of 0.1 units on the FFT graph. So, the noise level with the shutter closed must be less than 10^{-2} units in order to "see" virtually anything with the NSOM. If the noise is higher than 10^{-2} further shielding is required.

NOTE: Movement of people and objects in the vicinity of the NSOM make a tremendous difference in the amount of noise observed. If people (you included) and objects will be moved/moving during the experiment, be sure to simulate this while testing noise levels.

Sample Placement

The sample must be placed on a transparent (to NIR) support structure (slide or coverslip) and then placed directly over the InGaAs detector. The fiber is then brought in from the top (detector underneath). The fiber is held in one position and the detector and sample

are raster scanned during data collection. It is necessary attach the sample support structure (slide or coverslip) to the stage, otherwise the fiber may simply drag around the sample and inappropriate spectra will be collected.

General Scanning

Scanning may be done in several different ways. The first is a simple straight linear segment of a sample; we will call this 1-D scanning. The second method of scanning is an area of a sample or 2-D scanning. The third type of scanning which has not been implemented yet is to do 2-D scanning but vary the distance the probe tip is from the sample. This we will call 3-D scanning. It has been found that one directional scanning will not always produce reliable results. One directional scanning in 2-D scans is how they are typically done. This is described in the nsom2D1way.m file as:

```
% First a scan in the "X direction" is completed as follows:
% The NSOM's stage will begin at the home position of (beginx), (beginy).
% It will move from (beginx) to (endx) in increments of (stepx). It will
% then return to beginx and repeat at the next y value incremented by
% (stepy). The process with repeat until the final point of (endx), (endy)
% has been reached. At each position, the NSOM will record (datapoints)
% number of samples per second from (piezoport) for a duration of (time)
% in seconds.
```

To verify the results a second way scan has been added to the code. It is described in the nsom2D2way.m file as:

% Next a scan in the "Y direction" is completed as follows:

% The NSOM's stage will again begin from the home position of

% (beginx), (beginy). This time, it will move from (beginy) to (endy) in

% increments of (stepy). It will then return to beginy and repeat at the

% next x value incremented by (stepx). The process will repeat until the

% final point of (endx), (endy) has been reached. At each position, the

% NSOM will record (datapoints) number of samples per second from (piezoport)

% for a duration of (time) in seconds.

1-D Scanning

Since it does not make sense to scan two ways in a 1-D scan, this scan is done using a different controlling program than the typical 2-D scan. The controlling program nsom2D1way.m is used. To scan using this controlling file, type open nsom2D1way at the command line in MatLab and setup the program following the comments in the file.

2-D Scanning

The controlling program nsom2D2way.m is used to scan an area of a sample. To scan using this controlling file, type open nsom2D2way at the command line in MatLab and setup the program following the comments in the file.

Viewing the Results

A variety of graphs may be produced from the collected data. A file for creating these graphs is called nsom2dplot.m. Type open nsom2dplot at the command line in MatLab

and follow the directions in the comments for producing a variety of graphs from the data.

3-D Scanning

3-D Scanning has not been implemented yet, but could easily be implemented by adding a third dimension to the code in nsom2D2way.m or nsom2D1way.m. The resulting output data files would need to contain three dimensional arrays.

Appendix C Figures

Figure C.1: Beamline connection to the NSOM Box.

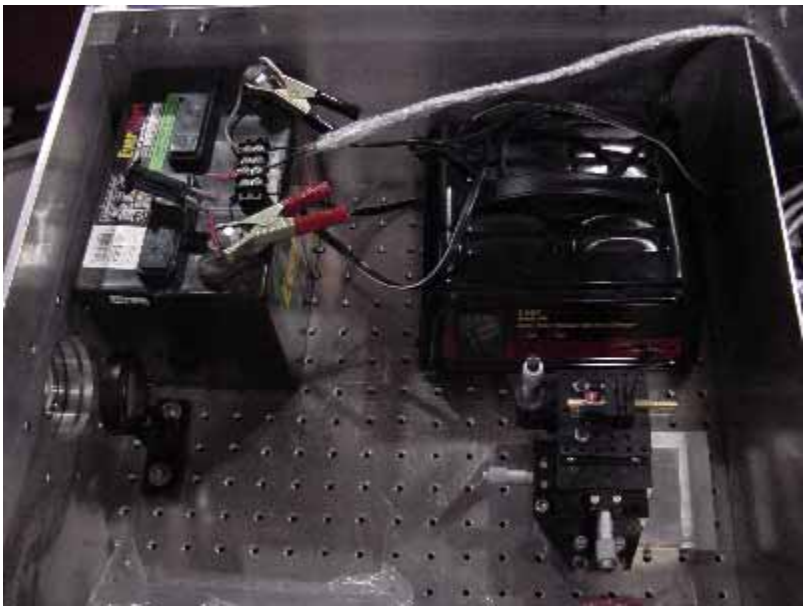

Figure C.2: Light coupling side of the NSOM box.

Figure C.3: Incoming beam coupled to the fiber with adjustments made to the 3D stage, fiber chuck, and the chuck mount.

Figure C.4: Detection side of the NSOM box showing fiber holder assembly, fiber, detector, and circuitry.

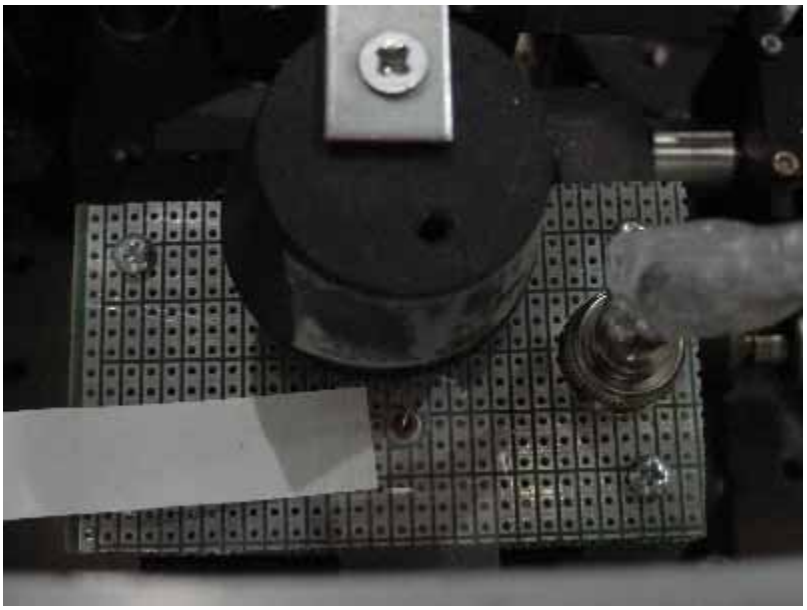

Figure C.5: Fiber, sample, and detector assembly.

Figure C.6: Aluminum foil shielding on all wires.

Figure C.7: WinRadio control panel.

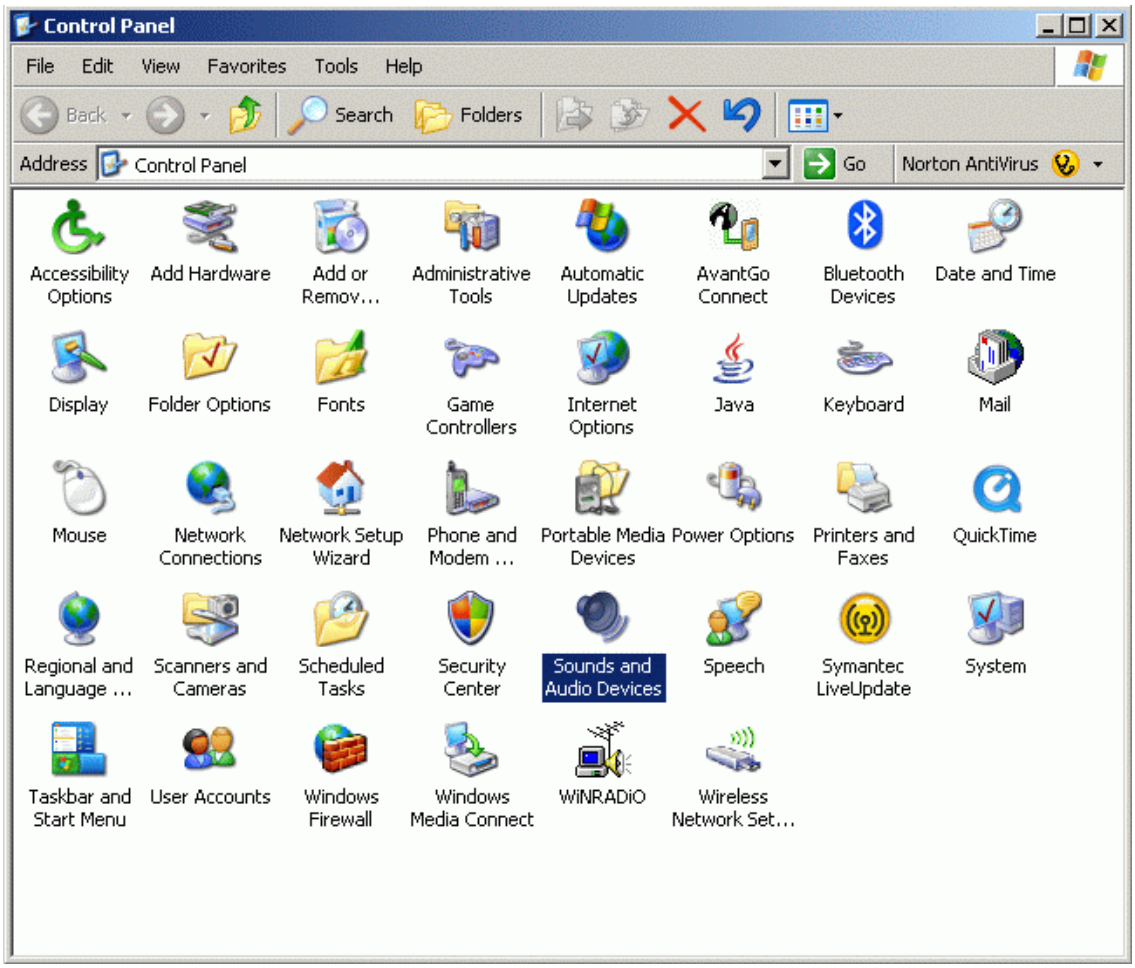

Figure C.8: Sounds and Audio Devices highlighted on the Control Panel window.

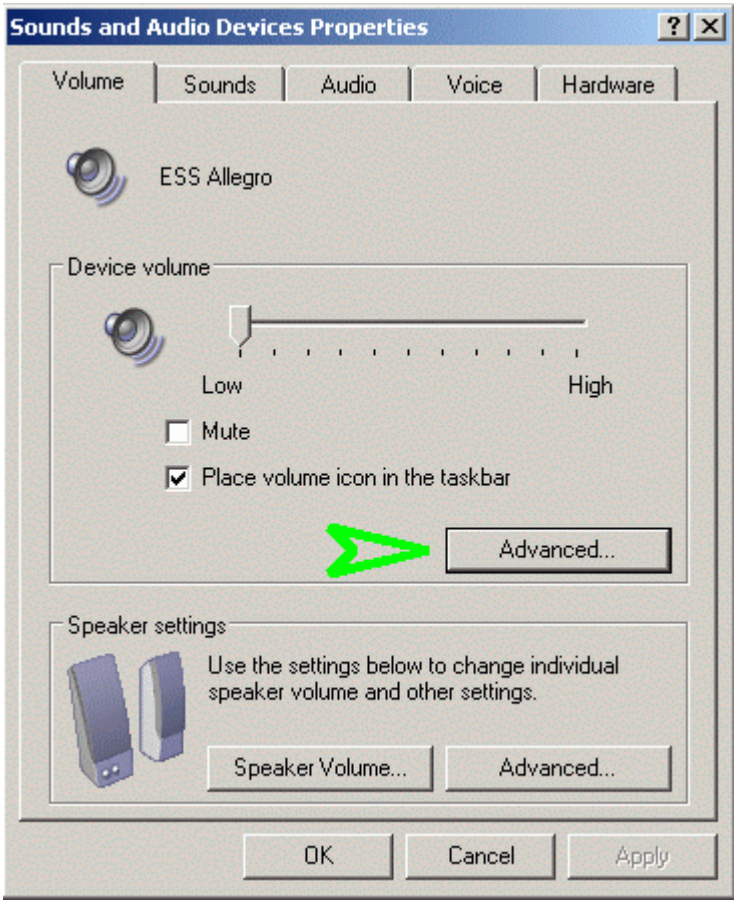

Figure C.9: Advanced button in the Sounds and Audio Devices Properties window.

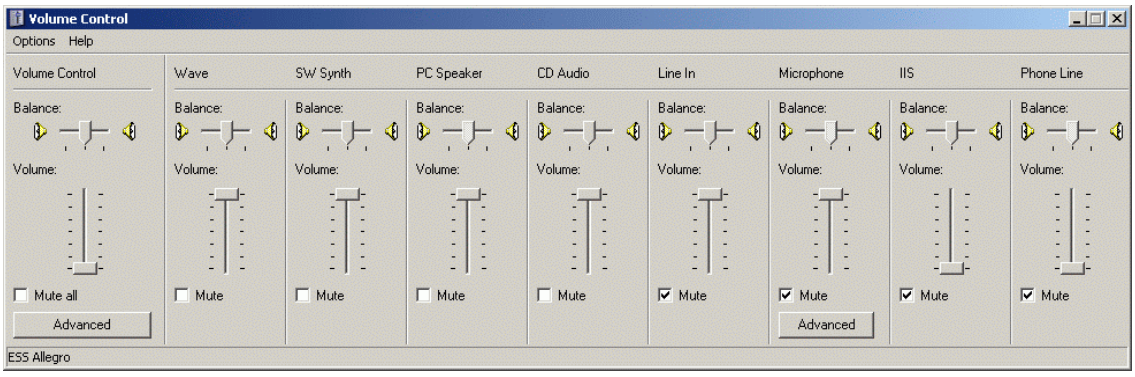

Figure C.10: Volume Control window.

Figure C.11: Speaker icon.

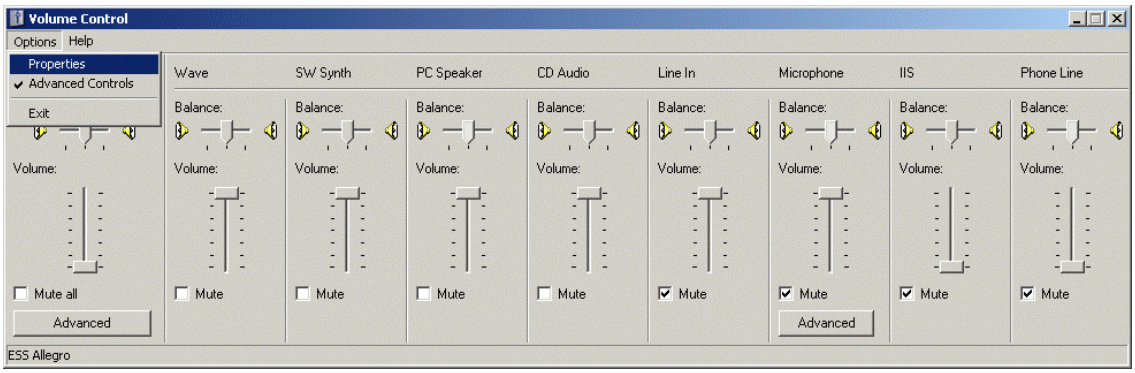

Figure C.12: Volume Control Options Menu showing Advanced Controls Checked.

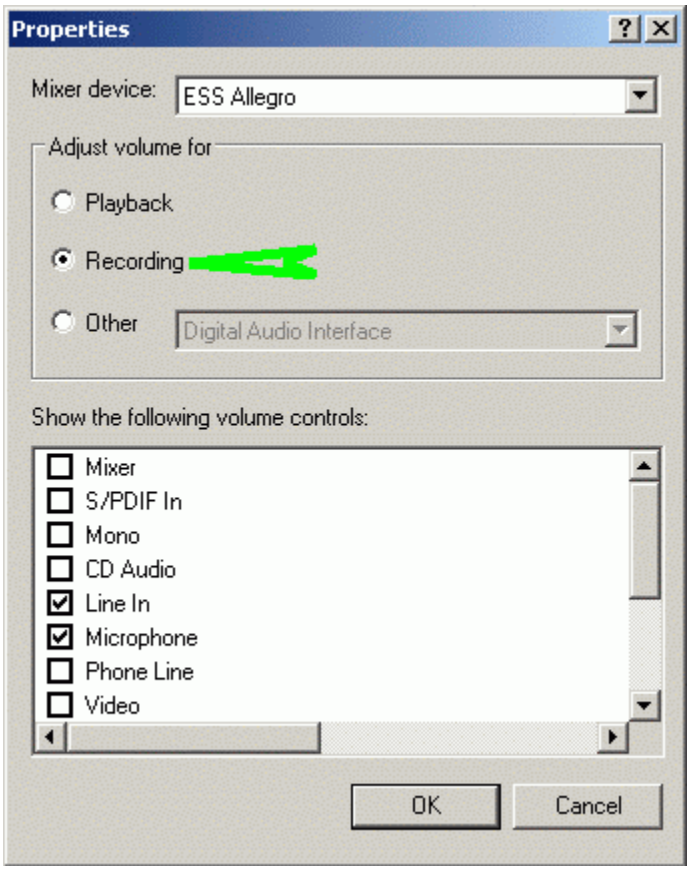

Figure C.13: Volume Control Properties showing Recording selected.

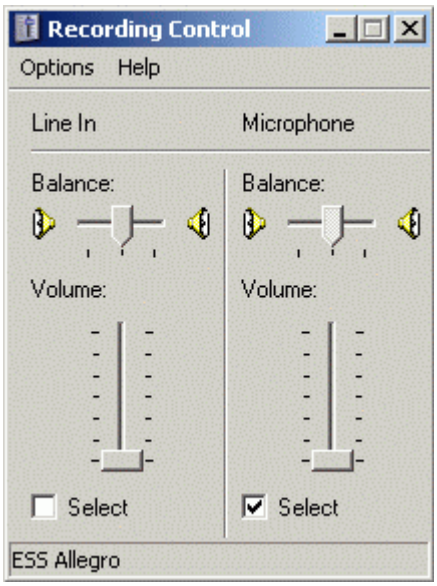

Figure C.14: Recording Control Window.

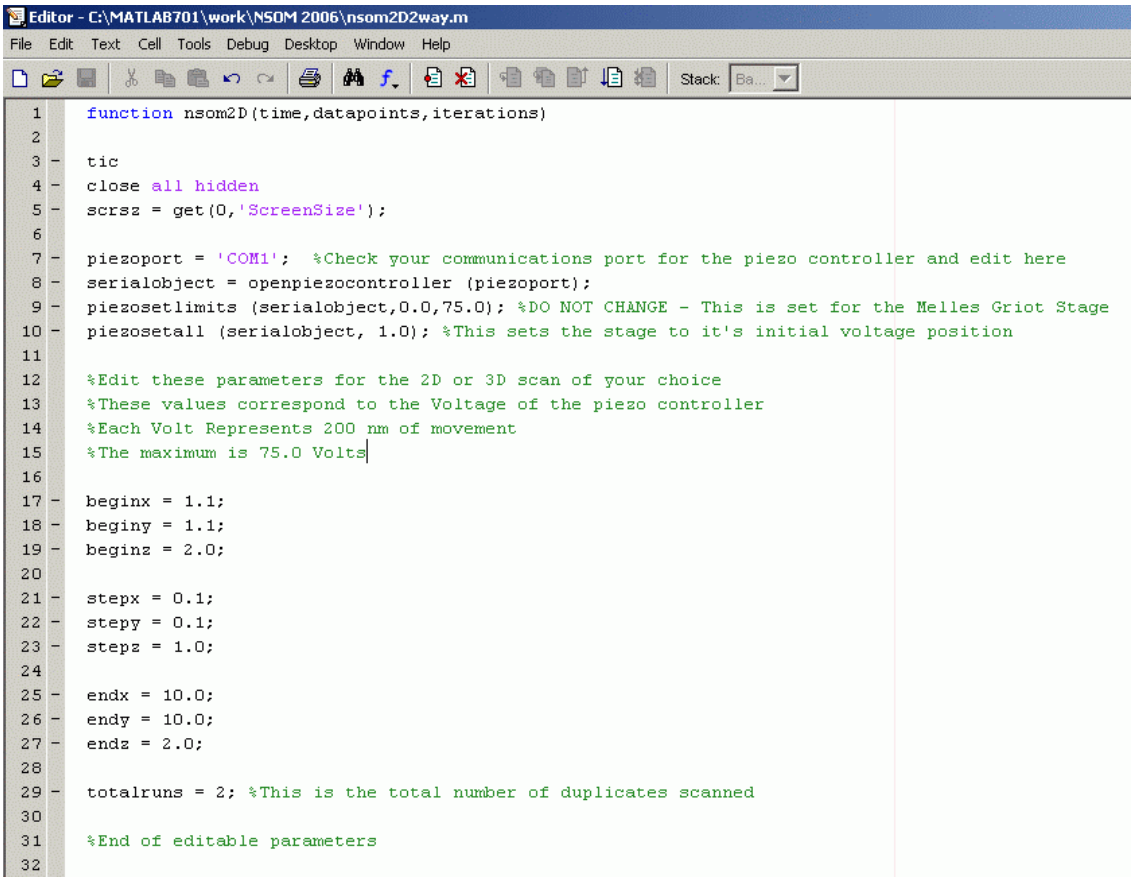

Figure C.15: nsom2D2way.m file in MatLab Editor window.

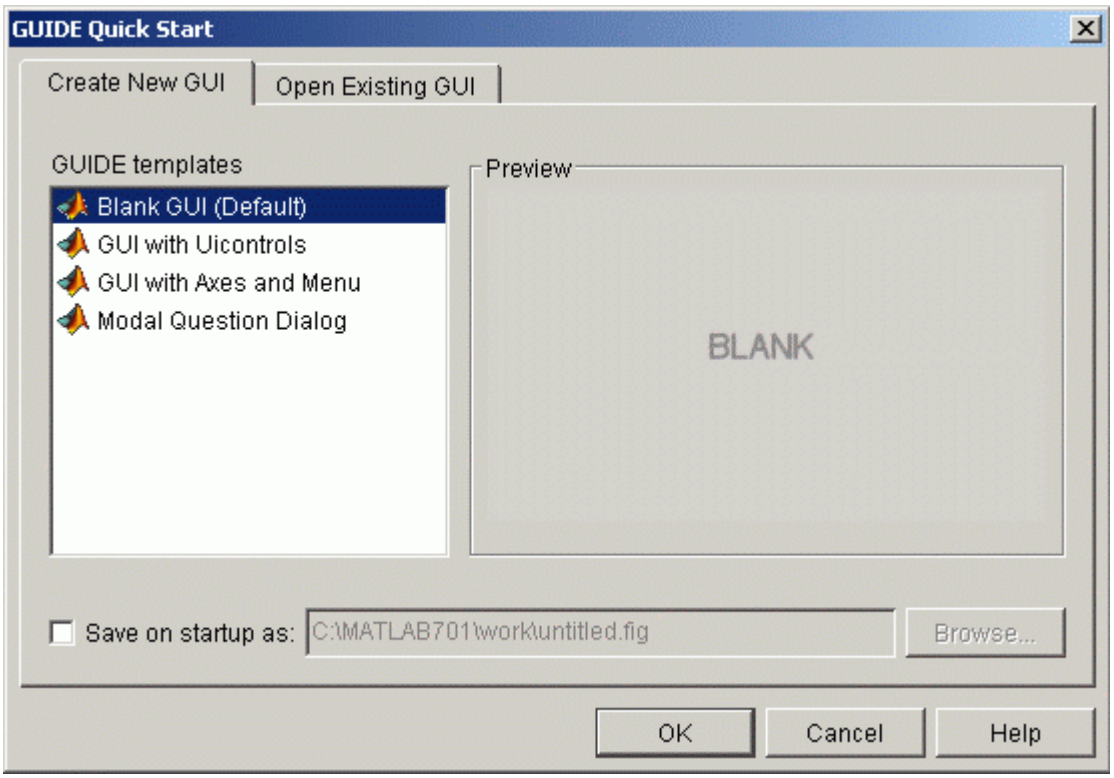

Figure C.16: GUIDE Quick Start window in MatLab.

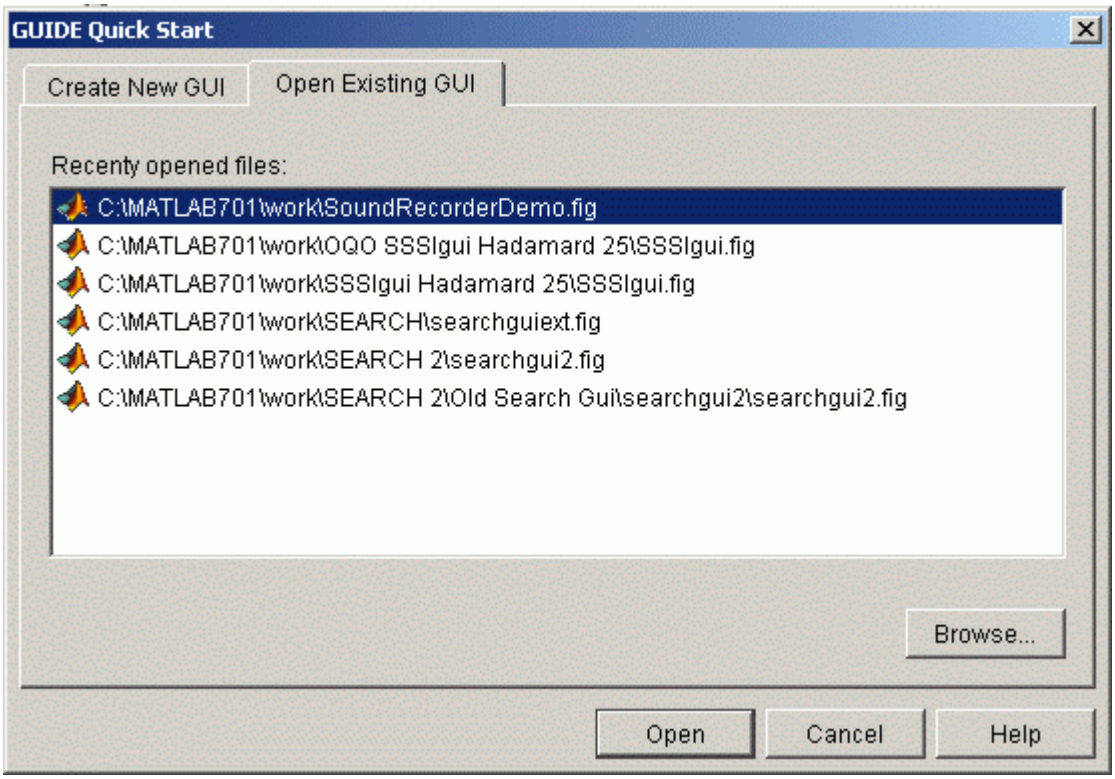

Figure C.17: GUIDE Quick Start window in MatLab with the Open Existing GUI tab selected.

Figure C.18: GUIDE initialization window.

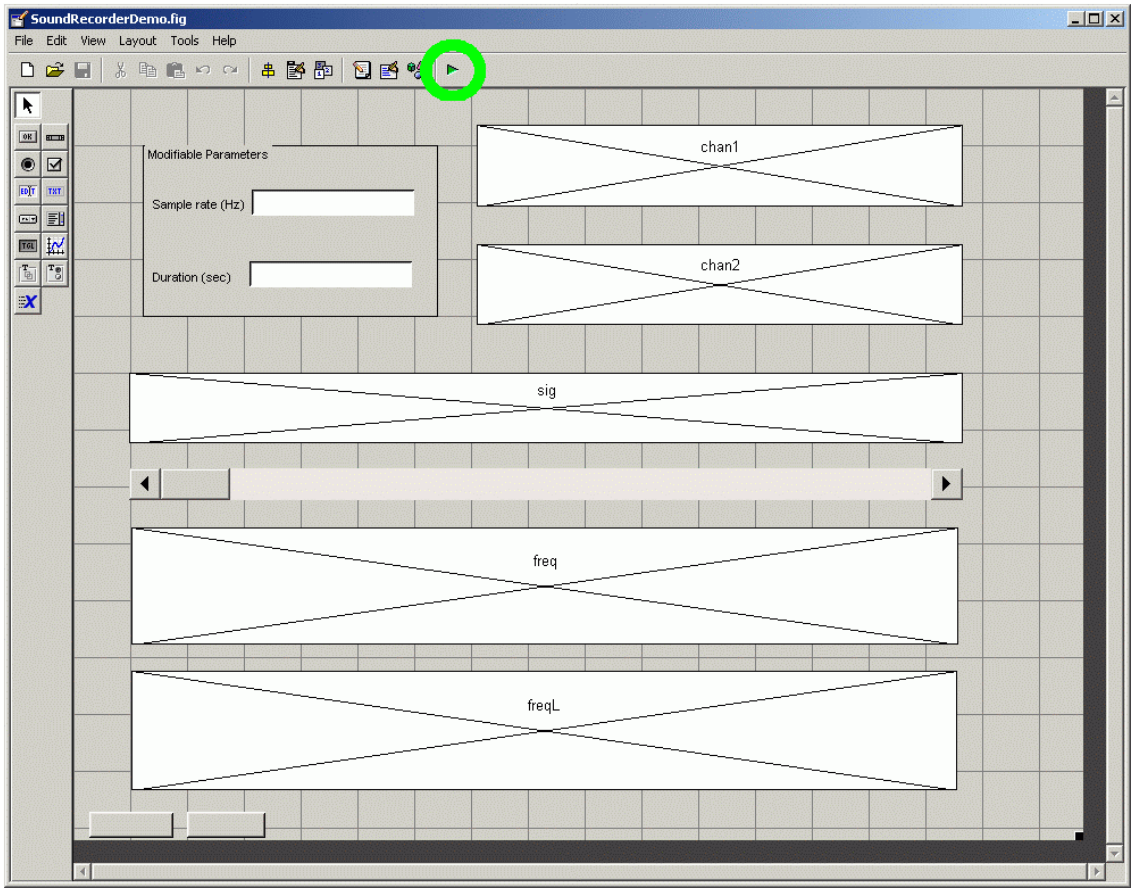

Figure C.19: SoundRecorderDemo.fig window with the play button circled in green.

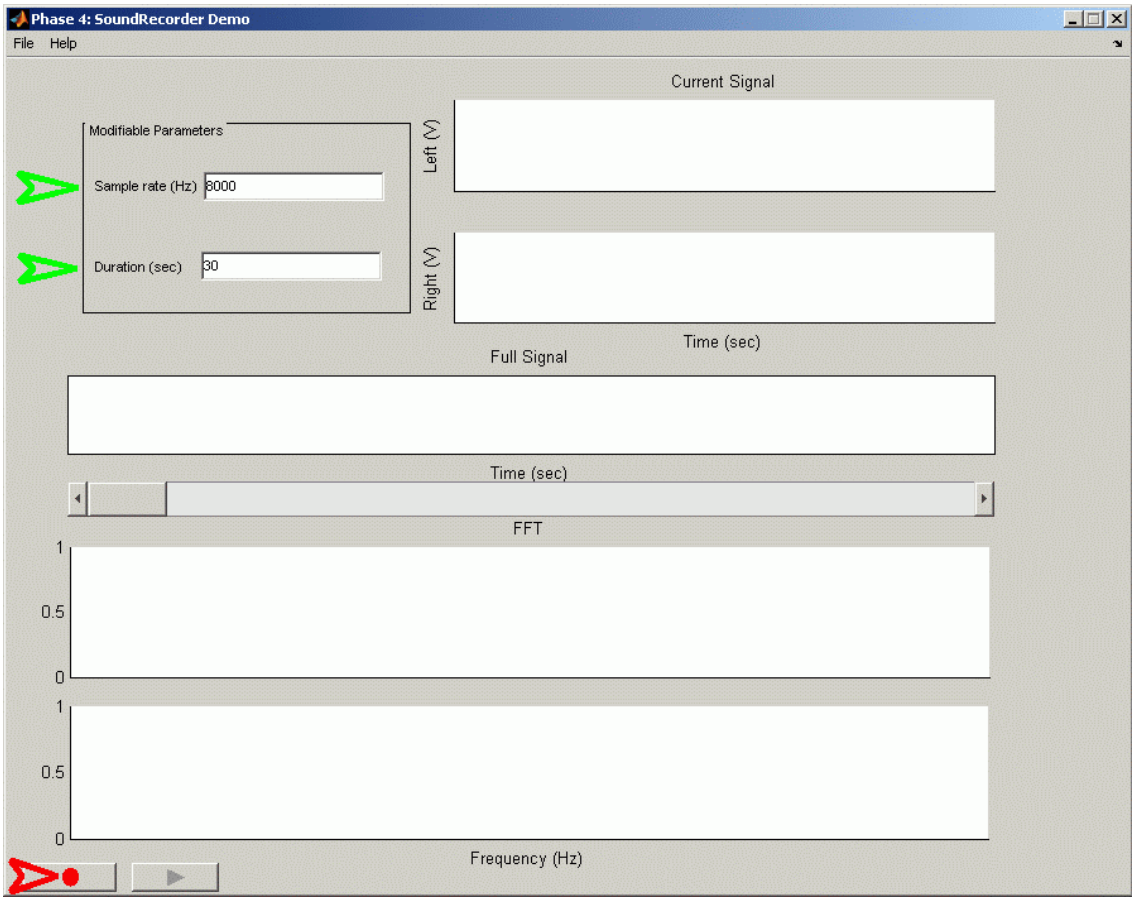

Figure C.20: SoundRecorder Demo GUI window with the Sample rate and Duration indicated by green arrows and the record button indicated by a red arrow.

References

1. Smith, R. B., *Introduction to Hyperspectral Imaging* MicroImages, Inc. : Lincoln, Nebraska 2006.
2. Skoog, D. A.; Holler, F. J.; Nieman, T. A., *Principles of Instrumental Analysis* 5th ed.; Harcourt Brace College Publishing 1998.
3. Hsu, S. M.; Burke, H.-h. K., Multisensor Fusion with Hyperspectral Imaging Data: Detection and Classification *Lincoln Laboratory Journal* **2003,** 14, (1), 145-159.
4. Norton, P., *Introduction to Computers*. 6 ed.; McGraw-Hill Technology Education: New York, 2006.
5. Manolakis, D.; Marden, D.; Shaw, G. A., Hyperspectral Image Processing for Automatic Target Detection Applications *Lincoln Laboratory Journal* **2003,** 14, 79-116.
6. Bianco, A. D.; Serafino, G.; Spock, G., An Introduction to Spectral Imaging *EuropastraBe* **2005,** 4, (A-9524), 1-36.
7. Asrar, G., *Theory and applications of optical remote sensing*. Wiley: New York, 1989; p xiv, 734.
8. Reeves, R. G.; American Society of Photogrammetry., *Manual of remote sensing*. 1st ed.; American Society of Photogrammetry: Falls Church, Va., 1975; p 2 v. (xxvi, 2144).
9. Fischer, M. M.; Scholten, H. J.; Unwin, D.; Foundation, E. S., *Spatial analytical perspectives on GIS*. Taylor & Francis: London ; Bristol, PA, 1996; p xvii, 256.
10. Richards, J. A., *Remote sensing digital image analysis : an introduction*. Springer-Verlag: Berlin ; New York, 1986; p xvii, 281.

11. Goetz, A. F. H.; Curtiss, B., Hyperspectral Imaging of the Earth: Remote Analytical Chemistry in an Uncontrolled Environment *Field Analytical Chemistry and Technology* **1996,** 1, (2), 67-76.
12. Burger, J.; Geladi, P., Hyperspectral NIR image regression part I: Calibration and correction *Journal of Chemometrics* **2005,** 19, 355-363.
13. Polder, G.; Heijden, G. v.; Keizer, L.; Young, I., Calibration and characterisation of imaging spectrographs. *J. of Near Infrared Spectroscopy* **2003,** 11, 193-210.
14. Gat, N., Imaging spectroscopy using tunable filters: a review *Proc. of SPIE* **2000,** 4056, 50-64.
15. Geladi, P.; Grahn, H.; NetLibrary Inc. *Multivariate image analysis*, Wiley: Chichester ; New York, 1996.
16. Wang, Y.; Veltkamp, D.; Kowalski, B., Multivariate instrument standardization *Anal. Chem.* **1991,** 63, 2750-2756.
17. Wang, Y.; Lysaght, M.; Kowalski, B., Improvement of multivariate calibration through instrument standardization *Anal. Chem.* **1992,** 64, 562-564.
18. Lodder, R. A.; Hieftje, G. M., Detection of Subpopulations in Near-Infrared reflectance Analysis. *Appl. Spectrosc.* **1988,** 42, (8), 1500-1512.
19. Lodder, R. A.; Hieftje, G. M., Quantile BEAST Attacks the False-Sample Problem in Near-Infrared Reflectance Analysis. *Appl. Spectrosc.* **1988,** 42, (8), 1351-1365.
20. Lodder, R. A.; Hieftje, G. M., Quantile Analysis: A Method for Characterizing Data Distributions. *Appl. Spectrosc.* **1988,** 42, (8), 1512-1520.
21. Bal, G., Radiative transfer equations with varying refractive index: a mathematical perspective. *J. Opt. Soc. Am. A* **2006,** 23, (7), 1639-1644.

22. Chandrasekhar, S., *Radiative transfer*. Dover Publications: New York,, 1960; p 393.

23. Ishimaru, A., *Wave propagation and scattering in random media*. Academic Press: New York, 1977; p v.

24. Olson, G. L.; Auer, L. H.; Hall, M. L., Di!usion, P1, and other approximate forms of radiation transport. *Journal of Quantitative Spectroscopy & Radiative Transfer* **2000,** 64, 619-634.

25. Turpault, R., A consistent multigroup model for radiative transfer and its underlying mean opacities. *Journal of Quantitative Spectroscopy & Radiative Transfer* **2005,** 94, 357–371.

26. Cox, A. N.; Allen, C. W., *Allen's Astrophysical quantities*. 4th ed.; AIP Press : Springer: New York, 2000; p xviii, 719.

27. Mihalas, D.; Weibel-Mihalas, B., *Foundations of radiation hydrodynamics*. Dover: Mineola, N.Y., 1999; p xv, 718.

28. Thomas, G. E.; Stamnes, K., *Radiative transfer in the atmosphere and ocean*. Cambridge University Press: Cambridge ; New York, 1999; p xxvi, 517 p.

29. Massart, D. L.; Vandeginste, B. G. M.; Deming, S. N.; Michotte, Y.; Kaufman, L., *Chemometrics: A Textbook*. Elsevier: Amsterdam, 1988; p 500.

30. Massart, D. L.; Vandeginste, B. G. M.; Buydens, L. M. C.; De Jong, S.; Lewi, P. J.; Smeyers-Verbeke, J., *Handbook of Chemometrics and Qualimetrics - Part A*. Elsevier: Amsterdam, 1997; p 886.

31. Massart, D. L.; Vandeginste, B. G. M.; Buydens, L. M. C.; De Jong, S.; Lewi, P. J.; Smeyers-Verbeke, J., *Handbook of Chemometrics and Qualimetrics - Part B*. Elsevier: Amsterdam, 1998; p 886.

32. Beebe, K. R.; Pell, R. J.; Seasholtz, M. B., *Chemometrics: A Practical Guide*. John Wiley & Sons, Inc.: New York, 1998.

33. Sharaf, M. A.; Illman, D. L.; Kowalski, B. R., *Chemometrics*. Wiley: New York, 1986.

34. Kowalski, B. R., *Chemometrics: Mathematics and Statistics in Chemistry*. Springer: New York, 2001.

35. Brereton, R. G., *Chemometrics: Data Analysis for the Laboratory and Chemical Plant*. John Wiley & Sons, Inc.: Chichester, UK, 2003.

36. Jolliffe, I. T., *Principal Components Analysis*. 2nd ed.; Springer-Verlag: NY, 2002; p 502.

37. Damiani, P. C.; Escandar, G. M.; Olivieri, A. C.; Goicoechea, H. C., Multivariate calibration: A powerful tool in pharmaceutical analysis. *Current Pharmaceutical Analysis* **2005,** 1, (2), 145-154.

38. Kalivas, J. H., Multivariate calibration, an overview. *Analytical Letters* **2005,** 38, (14), 2259-2279.

39. Soyemi, O.; Eastwood, D.; Zhang, L.; Li, H.; Karunamuni, J.; Gemperline, P.; Synowicki, R. A.; Myrick, M. L., Design and testing of a multivariate optical element: The first demonstration of multivariate optical computing for predictive spectroscopy. *Analytical Chemistry* **2001,** 73, (6), 1069-1079.

40. Nelson, M. P.; Aust, J. F.; Dobrowolski, J. A.; Verly, P. G.; Myrick, M. L., Multivariate optical computation for predictive spectroscopy. *Analytical Chemistry* **1998,** 70, (1), 73-82.

41. Myrick, M. L.; Soyemi, O. O.; Haibach, F.; Zhang, L.; Greer, A.; Li, H.; Priore, R.; Schiza, M. V.; Farr, J. R., Application of multivariate optical computing to near-infrared imaging. *Proceedings of SPIE-The International Society for Optical Engineering* **2002,** 4577, 148-157.

42. Myrick, M. L.; Soyemi, O. O.; Schiza, M. V.; Farr, J. R.; Haibach, F.; Greer, A.; Li, H.; Priore, R., Application of multivariate optical computing to simple near-infrared point measurements. *Proceedings of SPIE-The International Society for Optical Engineering* **2002,** 4574, 208-215.

43. Dai, B.; Urbas, A.; Douglas, C.; Lodder, R. A., Molecular Factor Computing for Predictive Spectroscopy. *Pharmaceutical Research* **2007,** In Press.

44. Dai, B. Simulations-Guided Design of Process Analytical Sensor using Molecular Factor Computing. University of Kentucky, 2007.

45. Lasch, P.; Wasche, W.; McCarthy, W. J.; Muller, G.; Naumann, D., FT-IR Microspectroscopic Imaging of Human Carcinoma Thin Sections by FT-IR Microspectrometry. *Spectroscopy of Biological Molecules: Modern Trends* **1997**, 441-442.

46. Cassis, L. A.; Dai, B.; Urbas, A.; Lodder, R. A., In vivo applications of a molecular computing-based high-throughput NIR spectrometer. *Proc. SPIE-Int. Soc. Opt. Eng.* **2004,** 5329, (239-253).

47. Mitchell, M. E.; Sidawy, A. N., The pathophysiology of atherosclerosis. *Semin Vasc Surg* **1998,** 11, (3), 134-41.

48. Ross, R., Atherosclerosis--an inflammatory disease. *N Engl J Med* **1999,** 340, (2), 115-26.

49. Ross, R., Atherosclerosis is an inflammatory disease. *Am Heart J* **1999,** 138, (5 Pt 2), S419-20.

50. Hamilton, C. A., Low-density lipoprotein and oxidised low-density lipoprotein: their role in the development of atherosclerosis. *Pharmacol Ther* **1997,** 74, (1), 55-72.

51. Nordestgaard, B. G.; Nielsen, L. B., Atherosclerosis and arterial influx of lipoproteins. *Curr Opin Lipidol* **1994,** 5, (4), 252-7.

52. Berliner, J.; Leitinger, N.; Watson, A.; Huber, J.; Fogelman, A.; Navab, M., Oxidized lipids in atherogenesis: formation, destruction and action. *Thromb Haemost* **1997,** 78, (1), 195-9.

53. Navab, M.; Berliner, J. A.; Watson, A. D.; Hama, S. Y.; Territo, M. C.; Lusis, A. J.; Shih, D. M.; Van Lenten, B. J.; Frank, J. S.; Demer, L. L.; Edwards, P. A.; Fogelman, A. M., The Yin and Yang of oxidation in the development of the fatty streak. A review based on the 1994 George Lyman Duff Memorial Lecture. *Arterioscler Thromb Vasc Biol* **1996,** 16, (7), 831-42.

54. Navab, M.; Hama, S. Y.; Nguyen, T. B.; Fogelman, A. M., Monocyte adhesion and transmigration in atherosclerosis. *Coron Artery Dis* **1994,** 5, (3), 198-204.

55. Schwartz, C. J.; Valente, A. J.; Sprague, E. A., A modern view of atherogenesis. *Am J Cardiol* **1993,** 71, (6), 9B-14B.

56. DiCorleto, P. E., Cellular mechanisms of atherogenesis. *Am J Hypertens* **1993,** 6, (11 Pt 2), 314S-318S.

57. Stary, H. C., The sequence of cell and matrix changes in atherosclerotic lesions of coronary arteries in the first forty years of life. *Eur Heart J* **1990,** 11 Suppl E, 3-19.

58. Fuster, V.; Moreno, P. R.; Fayad, Z. A.; Corti, R.; Badimon, J. J., Atherothrombosis and High-Risk Plaque Part I: Evolving Concepts. *J Am Coll Cardiol* **2005,** 46, (6), 937-954.

59. Shah, P. K., Mechanisms of Plaque Vulnerability and Rupture. *J Am Coll Cardiol* **2003,** 41, (4), 15S-22S.

60. Virmani, R.; Burke, A. P.; Andrew, F.; D., K. F., Pathology of the Unstable Plaque. *Prog Cardiovasc Dis* **2002,** 44, (5), 349-356.

61. Virmani, R.; Kolodgie, F. D.; Burke, A. P.; Farb, A.; Schwartz, S. M., Lessons From Sudden Coronary Death: A Comprehensive Morphological Classification Scheme for Atherosclerotic Lesions. *Arterioscler Thromb Vasc Biol* **2000,** 20, (5), 1262-1275.

62. Virmani, R.; Burke, A.; Kolodgie, F.; al., e., Pathology of the thin-cap fibroatheroma: a type of vulnerable plaque. *J Interv Cardiol* **2003,** 16, (3).

63. Kolodgie, F.; Burke, A.; Farb, A.; al., e., The thin-cap fibroatheroma: a type of vulnerable plaque: the major precursor lesion to acute coronary syndromes. *Curr Opin Cardiol* **2001,** 16, (5), 285-292.

64. Yan, W. D.; Perk, M.; Chagpar, A.; Wen, Y.; Stratoff, S.; Schneider, W. J.; Jugdutt, B. I.; Tulip, J.; Lucas, A., Laser-Induced Fluorescence .3. Quantitative-Analysis of Atherosclerotic Plaque Content. *Lasers in Surgery and Medicine* **1995,** 16, (2), 164-178.

65. Oraevsky, A. A.; Jacques, S. L.; Pettit, G. H.; Sauerbrey, R. A.; Tittel, F. K.; Nguy, J. H.; Henry, P. D., Xecl Laser-Induced Fluorescence of Atherosclerotic Arteries - Spectral Similarities between Lipid-Rich Lesions and Peroxidized Lipoproteins. *Circulation Research* **1993,** 72, (1), 84-90.

66. Laifer, L. I.; Obrien, K. M.; Stetz, M. L.; Gindi, G. R.; Garrand, T. J.; Deckelbaum, L. I., Biochemical Basis for the Difference between Normal and Atherosclerotic Arterial Fluorescence. *Circulation* **1989,** 80, (6), 1893-1901.

67. Rokosova, B.; Rapp, J. H.; Porter, J. M.; Bentley, J. P., Composition and Metabolism of Symptomatic Distal Aortic Plaque. *Journal of Vascular Surgery* **1986,** 3, (4), 617-622.

68. Smith, E. B., Influence of Age and Atherosclerosis on Chemistry of Aortic Intima .2. Collagen and Mucopolysaccharides. *Journal of Atherosclerosis Research* **1965,** 5, (2), 241-&.

69. Noble, N. L.; Boucek, R. J.; Kao, K. Y. T., Biochemical Observations of Human Atheromatosis - Analysis of Aortic Intima. *Circulation* **1957,** 15, (3), 366-372.

70. Campa, J. S.; Greenhalgh, R. M.; Powell, J. T., Elastin degradation in abdominal aortic aneurysms. *Atherosclerosis* **1987,** 65, (1-2), 13-21.

71. Powell, J.; Greenhalgh, R. M., Cellular, enzymatic, and genetic factors in the pathogenesis of abdominal aortic aneurysms. *J Vasc Surg* **1989,** 9, (2), 297-304.

72. Rizzo, R. J.; McCarthy, W. J.; Dixit, S. N.; Lilly, M. P.; Shively, V. P.; Flinn, W. R.; Yao, J. S., Collagen types and matrix protein content in human abdominal aortic aneurysms. *J Vasc Surg* **1989,** 10, (4), 365-73.

73. Sakalihasan, N.; Heyeres, A.; Nusgens, B. V.; Limet, R.; Lapiere, C. M., Modifications of the extracellular matrix of aneurysmal abdominal aortas as a function of their size. *Eur J Vasc Surg* **1993,** 7, (6), 633-7.

74. Baxter, B. T.; Davis, V. A.; Minion, D. J.; Wang, Y. P.; Lynch, T. G.; McManus, B. M., Abdominal aortic aneurysms are associated with altered matrix proteins of the nonaneurysmal aortic segments. *J Vasc Surg* **1994,** 19, (5), 797-802; discussion 803.

75. White, J. V.; Mazzacco, S. L., Formation and growth of aortic aneurysms induced by adventitial elastolysis. *Ann N Y Acad Sci* **1996,** 800, 97-120.

76. Ghorpade, A.; Baxter, B. T., Biochemistry and molecular regulation of matrix macromolecules in abdominal aortic aneurysms. *Ann N Y Acad Sci* **1996,** 800, 138-50.

77. White, J. V.; Haas, K.; Phillips, S.; Comerota, A. J., Adventitial elastolysis is a primary event in aneurysm formation. *J Vasc Surg* **1993,** 17, (2), 371-80; discussion 380-1.

78. Ernst, C. B., Abdominal aortic aneurysm. *N. Engl. J. Med.* **1993,** 328, 1167-1172.

79. Ciurczak, E. W.; Burns, D. A., *Handbook of Near-Infrared Analysis*. 2nd ed.; Marcel Dekker Inc.: New York, 2001; p 814.

80. O'Kelly, T. J.; Heather, B. P., General practice-based population screening for abdominal aortic aneurysms: a pilot study. *Br. J. Surg.* **1989,** 76, 305-310.

81. Lindholm, L.; Ejlertsson, G.; Forsberg, L.; Norgren, L., Low prevalence of abdominal aortic aneurysm in hypertensive patients. *Acta Med. Scand.* **1985,** 218, 305-310.

82. Harris, P., Screening for aortic aneurysm: the surgical perspective. *J. Med. Screening* **1994,** 1, 106-109.

83. Law, M. R.; Morris, J.; Wald, N. J., Screening for abdominal aortic aneurysms. *J. Med. Screening* **1994,** 1, 110-116.

84. Daugherty, A.; Cassis, L. A., Mouse Models of Abdominal Aortic Aneurysms. *Arterioscler. Thromb. Vasc. Biol.* **2004,** 24, (3), 429-434.

85. Wetzel, D. L.; Post, G. R.; Lodder, R. A., Synchrotron Infrared Microspectroscopic Analysis of Collagens I, III, and Elastin on the Shoulders of Human Thin-Cap Fibroatheromas. *Vib. Spectrosc.* **2005,** 38, 53-59.

86. Bialkowski, S. E., Species discrimination and quantitative estimation using incoherent linear optical signal processing of emission signals. *Analytical Chemistry* **1986,** 58, (12), 2561-2563.

87. Prakash, A. M. C.; Stellman, C. M.; Booksh, K. S., Optical regression: a method for improving quantitative precision of multivariate prediction with single channel spectrometers. *Chemometrics and Intelligent Laboratory Systems* **1999,** 46, (2), 265-274.

88. Valens, C., *The Hadamard Transform, http://perso.wanadoo.fr/polyvalens/clemens/transforms/hadamard/hadamard.html (25 Nov. 2005)*.

89. Douglas, C. C.; Harris, J. C.; Iskandarani, M.; Johnson, C. R.; Lodder, R. A.; Parker, S. G.; Cole, M. J.; Ewing, R.; Efendiev, Y.; Lazarov, R.; Qin, G. In *Dynamic Contaminant Identification in Water*, ICCS, 2006; Alexandrov, V. N.; Albada, G. D. v.; Sloot, P. M. A.; Dongarra, J. J., Eds. Springer-Verlag: 2006; pp 393-400.

90. Cassis, L. A.; Urbas, A.; Lodder, R. A., Hyperspectral Integrated Computational Imaging. *Anal. Bioanal. Chem.* **2005,** 382, 868-872.

91. Daugherty, A.; Manning, M. W.; Cassis, L. A., Angiotensis II promotes atherosclerotic lesions and aneurysms in apolipoprotein E-deficient mice. *J. Clin. Invest.* **2000,** 105, 1605-1612.

92. Urbas, A.; Manning, M. W.; Daugherty, A.; Cassis, L. A.; Lodder, R. A., Near-Infrared Spectrometry of Abdominal Aortic Aneurysm in the ApoE -/- Mouse. *Anal. Chem.* **2003,** 75, (14), 3650-3655.

93. Symons, W. C.; Whites, K. W.; Lodder, R. A., Theoretical and Experimental Characterization of Near-Field Scanning Microwave Microscope (NSMM). *IEEE T Microw Theory* **2003,** 51, (1), 91-99.

94. Balanis, C. A., *Advanced Engineering Electromagnetics*. Wiley: New York, 1989.

Vita

Justin Clay Harris
Born: August 9, 1979
Marion, Ohio

EDUCATION

The Ohio State University Columbus, OH June 2003

B.S. – Chemistry *(Cum Laude with Honors and with Distinction in Analytical Chemistry)*
Honors Thesis: Cooperative Hydrogen Bonding in Supercritical Fluids
(Dr. Susan V. Olesik)

PROFESSIONAL

University of Kentucky Lexington, KY May 2004 - Aug 2006
- Research Assistant in Analytical Division, Department of Chemistry

The Ohio State University Columbus, OH Sept 2002 - June 2003
- Research Assistant in Analytical Division, Department of Chemistry

University of Kentucky Lexington, KY Aug 2003 - May 2004
- Teaching Assistant in Analytical and Physical Divisions, Department of Chemistry

HONORS

- Jeffrey Research Fellowship, University of Kentucky, 2005 – 2006
- Gill Heart Institute Cardiovascular Research Day Award for Excellence in Scientific Presentation, 2005
- Paul Murrill Fellowship, University of Kentucky, 2003 – 2005
- Honors Research Fellowship, The Ohio State University, 2002 –2003
- ACS Division of Analytical Chemistry 2002 Undergraduate Award
- 2000 CRC Press LLC Freshman Chemistry Achievement Award
- Energy Research Undergraduate Laboratory Fellowship (ERULF), Oak Ridge National Laboratories / Department of Energy (DOE), 2000

PUBLICATIONS

Harris, J. C.; Butcher, J. T.; Lodder, R. A. Hyperspectral Imaging Calibration. In *Linear Regression Modeling.* Elsevier. In Progress to be submitted October, 2007.

Harris, J. C.; Lodder, R. A. Improved Bioinformatics Using Cluster Analysis with Quantile-Quantile Plots and Synchrotron IR Microspectrometry for the Study of Abdominal Aortic Aneurysm. In Progress to be submitted to Analytical Chemistry October, 2007.

Harris, J. C.; Lodder, R. A. Integrated Computational Imaging with a Near-Infrared Near-field Scanning Optical Microscope (ICI NIR-NSOM). In Progress to be submitted October, 2007.

Douglas, C. C.; Harris, J. C.; Iskandarani, M.; Johnson, C. R.; Lodder, R. A.; Parder, S.; Cole, M. J.; Ewing, R.; Efendiev, Y.; Lazarov, R.; Qin, G. Dynamic Contaminant Identification in Water. In *ICCS, Part III, LNCS 3993;* V.NAlexandrov et al. (Eds.); Springer-Verlag: Berlin Heidelberg, 2006; pp 393-400.

Harris, J. C. Cooperative Hydrogen Bonding in Supercritical Fluids. B.S. Honors Thesis, The Ohio State University, Columbus, OH, 2003.

Harris, J. C.; Datskos, P. Enhanced Sensitivity of Micro Mechanical Chemical Sensors through Structural Variation; Report P00-109372; Oak Ridge National Laboratory: Oak Ridge, TN, 2001.

PRESENTATIONS

Harris, J.C.; Police, S. B.; Lodder, R. A.; Cassis, L. A. Comparison of the Effects of Diets Enriched in Sucrose Versus D-Tagatose on Obesity, Hypercholesterolemia and Atherosclerosis in LDL Receptor Deficient Mice. Presented at the Gill Heart Institute Cardiovascular Research Day, Lexington, Kentucky, 2006.

Harris, J. C.; Cassis, L. A.; Lodder, R. A. Synchrotron IR Analysis of Murine Abdominal Aortic Aneurysm. Presented at the 57th Annual Pittsburgh Conference on Analytical Chemistry and Applied Spectroscopy, Orlando, FL, 2006.

Harris, J. C.; Dolan, R. E.; Lodder, R. A. Near-Infrared Near-Field Scanning Optical Microscopy (NIR-NSOM) of Carbon Nanotubes. Presented at the 57th Annual Pittsburgh Conference on Analytical Chemistry and Applied Spectroscopy, Orlando, FL, 2006.

Harris, J. C.; Fackler, J.; Lodder, R. A. Improved Bioinformatics Using Synchrotron IR Microspectrometry for the Study of Abdominal Aortic Aneurysm. Presented at the Gill Heart Institute Cardiovascular Research Day, Lexington, Kentucky, 2005.

Harris, J. C.; Fackler, J.; Lodder, R. A. Subwavelength Spectrometric Imaging and Molecular Computing in Studies of Atherosclerosis and Aneurysm. Presented at the Gill Heart Institute Cardiovascular Research Day, Lexington, Kentucky, 2005.

Harris, J. C.; Wetzel, D. L.; Lodder, R. A. Integrated Computational Imaging with a Near-Infrared Near-field Scanning Optical Microscope. Presented at the 3rd International Conference on Advanced Vibrational Spectroscopy, Delavan, Wisconsin, 2005.

Harris, J. C.; Wetzel, D. L.; Lodder, R. A. Arterial Wall Thickening in the ApoE -/- Mouse in the Absence of Aneurysm. Presented at the 3rd International Conference on Advanced Vibrational Spectroscopy, Delavan, Wisconsin, 2005.

Harris, J. C.; Hatcher, J.; Lodder, R. A. Solid State Spectral Imaging for Rapid Analysis in Diverse Environments. Presented at the 31st Annual Naff Symposium, Lexington, KY, 2005.

Lodder, R. A.; Harris, J. C. Scanning For Extinct Astrobiological Residues and Current Habitats (SEARCH) Using Integrated Computational Imaging. Presented at the 56th Annual Pittsburgh Conference on Analytical Chemistry and Applied Spectroscopy, Orlando, FL, 2005.

Lodder, R. A.; Harris, J. C. NIR-NSOM Using Hyperspectral Integrated Computational Imaging. Presented at the 56th Annual Pittsburgh Conference on Analytical Chemistry and Applied Spectroscopy, Orlando, FL, 2005.

Harris, J. C.; Datskos, P. Enhanced Sensitivity of Micro Mechanical Chemical Sensors through Structural Variation. Presented at the 15th Annual National Conference of Undergraduate Research, Lexington, KY, 2001.